iWay

iWay Transformer User's Manual
Version 5 Release 5 Service Manager (SM)

DN3501748.1006

Preface

This document is written for system integrators who require data transformations. It describes how to use the iWay Transformer to create transformations. A transformation is defined as the conversion of an input file of one data format to an output file of another data format. iWay Transformer uses an XML configuration file, called a template, to perform the actual transformation. It is assumed that readers understand system integration, data formats, and have a general understanding of Microsoft Windows and UNIX systems.

How This Manual Is Organized

The following table lists the numbers and titles of the chapters and appendixes for this manual with a brief description of the contents of each chapter and appendix.

Chapter/Appendix		Contents
1	Introducing iWay Transformer	Provides an overview of the iWay Transformer.
2	Getting Started	Describes the menus and options of the iWay Transformer graphical user interface.
3	Working with Projects	Describes how to design and work with transformation projects.
4	Configuring Input Items	Describes how to configure input items using iWay Transformer.
5	Configuring Output Items	Describes how to configure output items using iWay Transformer.
6	About Functions	Describes how to use functions in transformation projects.
A	Pre-Defined Functions	Describes how to use pre-defined functions in transformation projects.
B	Using Dictionary Files	Explains how to use dictionary files in EDI and SWIFT transformation projects.
C	Sample Transformations	Provides transformation tutorials using iWay Transformer that demonstrate looping and context techniques. In addition, information concerning embedded iWay response data is provided.

Documentation Conventions

The following table lists and describes the conventions that apply in this manual.

Convention	Description
`THIS TYPEFACE` or `this typeface`	Denotes syntax that you must enter exactly as shown.
`this typeface`	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
underscore	Indicates a default setting.
this typeface	Represents a placeholder (or variable), a cross-reference, or an important term.
this typeface	Highlights a file name or command.
Key + Key	Indicates keys that you must press simultaneously.
`{ }`	Indicates two or three choices; type one of them, not the braces.
|	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
`...`	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (…).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

To view a current listing of our publications and to place an order, visit our World Wide Web site, http://www.iwaysoftware.com. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about the iWay Transformer?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your iWay Transformer questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, http://www.informationbuilders.com. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.informationbuilders.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Help Us to Serve You Better

To help our consultants answer your questions effectively, please be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the environment information our consultants require.

Platform	
Operating System	
OS Version	
JVM Vendor	
JVM Version	

The following table lists the deployment information our consultants require.

Adapter Deployment	For example, JCA, Business Services Engine, iWay Service Manager
Container	For example, WebSphere
Version	
Enterprise Information System (EIS) - if any	
EIS Release Level	
EIS Service Pack	
EIS Platform	

The following table lists iWay related information needed by our consultants.

iWay Adapter	
iWay Release Level	
iWay Patch	

The following table lists the types of Application Explorer. Specify the version (and platform, if different than listed previously) in the columns provided.

Application Explorer Type	Version	Platform
Swing		
Servlet		
ASP		
Embedded in Service Designer		

The following table lists additional questions to help us serve you better.

Request/Question	**Error/Problem Details or Information**
Did the problem arise through a service or event.	
Provide usage scenarios or summarize the application that produces the problem.	
When did the problem start?	
Can you reproduce this problem consistently?	
Describe the **problem**.	
Describe the **steps** to reproduce the problem.	
Specify the **error** message(s).	
Any **change in the application environment**: software configuration, EIS/ database configuration, application, and so forth?	
Under what circumstance does the problem *not* occur?	

The following table lists error/problem files that might be applicable.

Input documents (XML instance, XML schema, non-XML documents)
Transformation files
Error screen shots
Error output files
Trace files
Service Manager package to reproduce problem
Custom functions and agents in use

Diagnostic Zip
Transaction log

See the *iWay Service Manager User's Guide* for information on tracing.

User Feedback

In an effort to produce effective documentation, the Documentation Services staff welcomes your opinions regarding this manual. Please use the Reader Comments form at the end of this manual to communicate suggestions for improving this publication or to alert us to corrections. You also can go to our Web site, http://www.iwaysoftware.com and use the Documentation Feedback form.

Thank you, in advance, for your comments.

iWay Software Training and Professional Services

Interested in training? Our Education Department offers a wide variety of training courses for iWay Software and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site, http://www.iwaysoftware.com or call (800) 969-INFO to speak to an Education Representative.

Interested in technical assistance for your implementation? Our Professional Services department provides expert design, systems architecture, implementation, and project management services for all your business integration projects. For information, visit our World Wide Web site, http://www.iwaysoftware.com.

Contents

CHAPTER 1

Introducing iWay Transformer

Topics:

- About iWay Transformer
- Supported Data Formats
- Using Help and Wizards

iWay Transformer is a rules-based, data transformation tool that converts an input file of one data format to an output file of another data format. The easy-to-use graphical user interface enables you to design transformation projects specific to your requirements.

This section introduces you to the function and features of iWay Transformer.

About iWay Transformer

iWay Transformer is an independent Java application that only requires a Java SDK to be installed. The Windows version is installed with iWay 5.5 SM. For more information on installing and configuring iWay Transformer, see the *iWay 5.5 SM Installation and Configuration* documentation.

iWay Transformer is used to transform a data structure and format to another data structure and format using rules defined by a developer. iWay Transformer can transform XML and non-XML input and output document types.

The main components of iWay Transformer are:

- **Transform Designer**, the Graphical User Interface (GUI) used to map a data transformation graphically.
- **Transformation Engine**, hosted by iWay Service Manager and responsible for performing the data transformation. The Transformation Engine is a robust, high performance, multi-threaded, and multi-process execution environment. It is based on a Java software platform that eliminates the requirement for custom programming of point-to-point interfaces between disparate systems.

  Note: In this document, iWay Transformer and the Transform Designer GUI component of iWay Transformer are synonymous.

iWay Transformer builds a transformation scenario within the framework of a project. Therefore, you create a transformation project for each conversion you want to perform. During the iWay Transformer installation, several directories are created to hold your project files and to provide sample transformation projects.

A project is made up of:

- A **template** file, an XML configuration file that contains the rules of the transformation. iWay Transformer provides sample templates that include the most common transformations. You can also create your own custom template.
- **Input files** are data format files and data structure files that specify the format and structure of the data source you are transforming.
- **Output files** are data format files and data structure files that specify the format and structure of the data source resulting from the transformation.
- **Functions** are written in Java code and perform calculations and manipulations of an input item. iWay Transformer provides several pre-defined functions, and you can also define your own.

After you set up the components of the project, you run the transformation. The transformation processes the input data according to the template file rules and parameters and produces the output file. You can view the results of the transformation and then save the results to a file.

Supported Data Formats

iWay Transformer can transform files *from* the following data formats:

- CDF (Column Defined Format)
- CSV (Comma Separated Values)
- EDI HIPAA (EDI-based for health care)
- EDI (Electronic Data Interchange)
  - EDI.X12
  - EDIFACT
- Fixed Width
- HL7 (Healthcare Level 7)
- IDoc (SAP Intermediate Document)
- SWIFT (International Financial)
- XML
- iWay XML Response Document

iWay Transformer can transform files *to* the following data formats:

- CDF (Column Defined Format)
- CSV (Comma Separated Values)
- EDI HIPAA (EDI-based for Healthcare)
- EDI (Electronic Data Interchange)
  - EDI.X12
  - EDIFACT
- Fixed Width
- HL7 (Healthcare Level 7)
- HTML
- IDoc (SAP Intermediate Document)

- SWIFT (International Financial)
- XML
- iWay XML Embedded Request
- iWay XML Request

Using Help and Wizards

iWay Transformer provides two wizards and a comprehensive online Help system to assist you in designing and performing transformations.

The iWay Transformer wizards are:

- **Project Wizard**, which guides you through the process of creating a simple transformation project.
- **Mapping Builder,** which guides you through the process of creating a function.

You can access online Help for the iWay Transformer GUI. Help provides detailed procedures and information related to all aspects of creating a project and performing a transformation.

CHAPTER 2

Getting Started

Topics:

- Starting iWay Transformer
- Using the Menu Bar and Toolbar
- Working With Project Views
- Customizing the User Interface
- Adjusting Project Properties
- Configuring XML Namespaces
- Adding Test Profiles
- Managing Server Connections
- Using the Find Tool
- Customizing Java Properties

The iWay Transformer user interface provides you with all the tools you require to create and work with transform projects. You are able to view and work with high-level and detailed aspects of a project. You can also customize the iWay Transformer interface to fit your requirements.

This section describes the menus and options available in the iWay Transformer user interface. You can learn about using the interface to work with transform projects in Chapter 3, *Working with Projects*.

Starting iWay Transformer

You can start iWay Transformer from the Start menu on Windows or by running the transformer shell script on UNIX.

Procedure: How to Start iWay Transformer

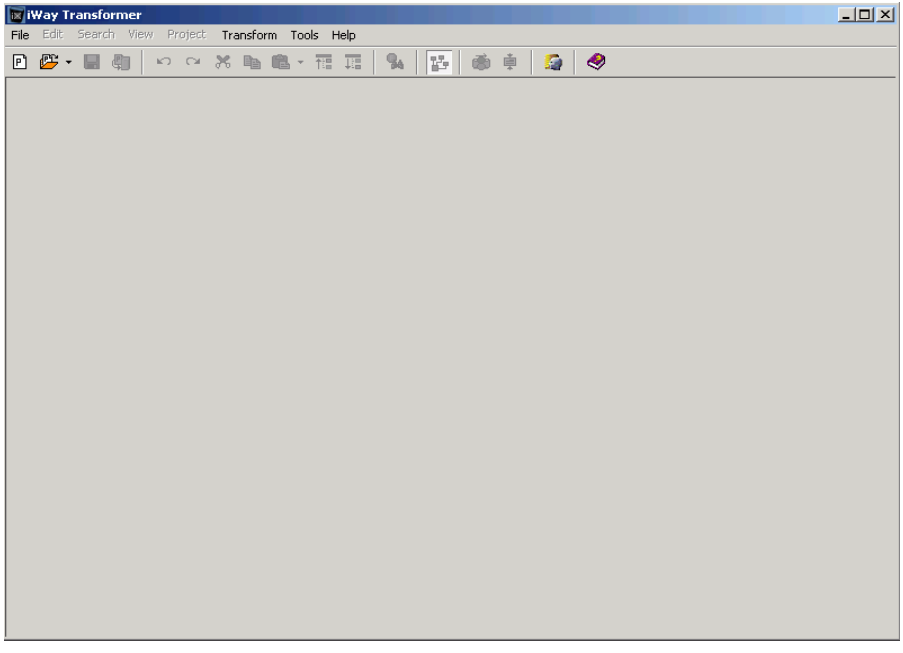
To start iWay Transformer:

1. On UNIX, execute the following:

   ```
   /opt/iWay55sm/tools/transformer/bin/transformer.sh
   ```

   or on Windows, from the Start menu, select *Programs, iWay 5.5 SM, tools*, and then *iWay Transformer*.

   The iWay Transformer window opens as shown in the following image.



2. From the File menu, choose one of the following options to work on a transformation project:

   New. Creates a new project without using the Project Wizard.

   Project Wizard. Provides a wizard interface to guide you in creating a new project.

   Open Project From File. Enables you to open an existing project from a local file system.

Open Project From Server. Enables you to open an existing project that resides in iWay Service Manager.

Recent Files. Provides a list of recently used project files.

Template. Enables you to import an existing iWay Transformer template (.xch) file.

Using the Menu Bar and Toolbar

To view the available menus and options in iWay Transformer, use the XML_to_HTML_Table.gxp sample project supplied with the software. The iWay Transformer toolbar contains buttons that enable you to perform many of the functions also available from the menu bar. For more information on the tool bar, see *iWay Transformer Toolbar Buttons* on page 2-11.

Procedure: How to Open the XML_to_HTML_Table Sample Project

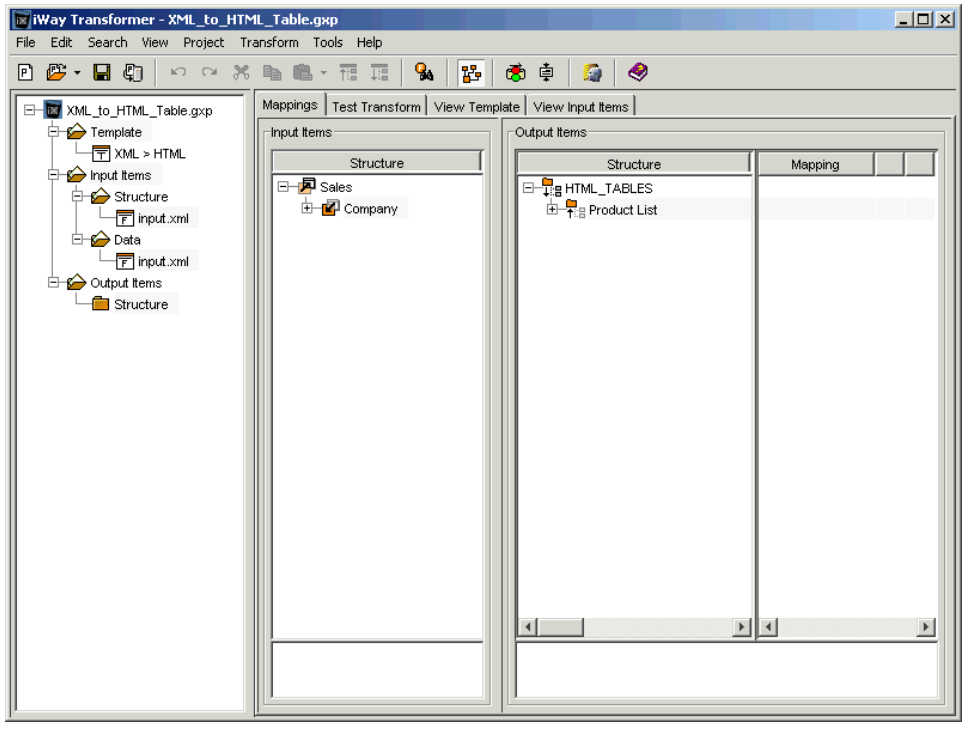
To open the XML_to_HTML_Table sample project:

1. Start iWay Transformer as described in *Starting iWay Transformer* on page 2-2.
2. From the File menu, select *Open Project* and then, *From File*.
3. In the Open dialog box, browse to the *XML_ to_ HTML_Table.gxp* file, which is located in the xml directory, under the sample_projects directory. The default path is:

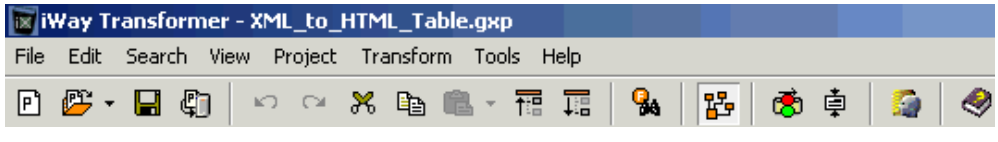
   ```
   C:\Program Files\iway55sm\tools\transformer\samples\
   sample_ projects\xml\XML_to_HTML_Table.gxp
   ```

4. Click *Open*.

The project opens in the default view, displaying the project pane on the left and the following tabs in the designer area on the right: Mappings, Test Transform, and View Template as shown in the following image.



The iWay Transformer menu bar and toolbar appear across the top of the window as shown in the following image. Most menu items have an equivalent button in the toolbar for quick access.



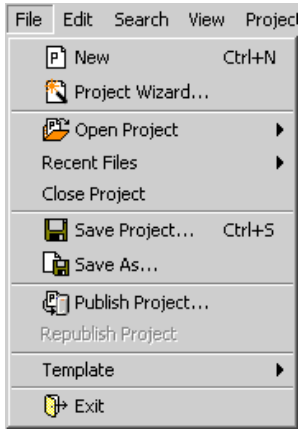
Reference: iWay Transformer Menu Bar

The following table lists and describes the options on the menu bar.

Option	Description
File	Enables you to manipulate, save, and close a project, to import and deploy new templates, and to exit iWay Transformer.
Edit	Provides standard editing tools for working with project elements in the Input and Output Items Structure panes.
Search	Provides the Find tool for locating specific information within a project.
View	Enables you to view various aspects of a project or all information related to the project.
Project	Provides menu access to configure the input and output items associated with a project. Enables you to access the Project Properties dialog box, where you can configure options to design and customize your project.
Transform	Provides options for configuring a transform.
Tools	Enables you to access Server Manager and customize the user interface.
Help	Enables you to access release information about iWay Transformer and online documentation.

Reference: File Menu

The following image shows the File menu options that become available after you select File from the menu bar.



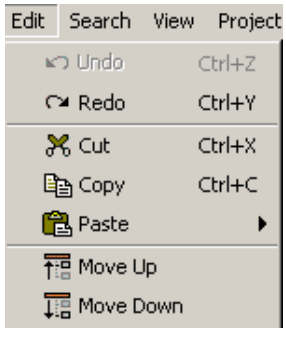
The following table lists and describes the options on the File menu.

Option	Description
New	Starts a new project by opening a new, untitled skeleton project that contains template, input item, and output item directories.
Project Wizard	Starts a new project using the Project Wizard, which assists you to create a project by performing the following four steps: **1.** Specify the project location. **2.** Specify the project type. **3.** Specify the structure of the input data and an input data file that contains sample data to model the transform. **4.** Specify output properties such as the XML, DTD, or XSD to represent the structure of the XML output data and options to configure XML output data of the transformation at run time.
Open Project	Displays a menu that enables you to open an existing project from a file residing on disk or iWay Server Manager Registry. Both options have dialog boxes associated with them from which you select the project you want to open. For more information on managing repositories, see *Managing Server Connections* on page 2-34.
Recent Files	Enables you to select and open a project from a list of projects on which you previously worked.
Close Project	Closes the current project. If you have not recently saved your work, iWay Transformer prompts you do so before closing.
Save Project	Saves changes made to a project. The project remains open.
Save As	Enables you to save and name your project. The project remains open.
Publish Project	Enables you to publish a transform into a server registry or configuration by specifying the name and registry or configuration location in addition to other properties such as the encoding.
Republish Project	Enables you to republish a transform into the former location within the server registry or configuration. The project must be opened from the server originally for this option to be enabled.

Option	Description
Template	Enables you to import or export a template as an iWay Transformer template (.xch) file. To manage your transformations, iWay Software recommends using projects, not templates.
Exit	Exits the iWay Transformer application. If a project is still open, iWay Transformer asks if you want to save the project before exiting.

Reference: Edit Menu

The following image shows the Edit menu options that become available after you select Edit from the menu bar.

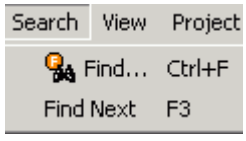


The following table lists and describes the options on the Edit menu.

Option	Description
Undo	Reverses the last edit.
Redo	Repeats the last edit.
Cut	Removes the selected item.
Copy	Copies the selected item.
Paste	Pastes the cut or copied item to a new location.
Paste Input	Available from Paste submenu. Pastes an input item into a subtree or node.
Paste Output	Available from Paste submenu. Pastes an output item into a subtree or node.
Move Up	Moves the selected item up on the structure tree.
Move Down	Moves the selected item down on the structure tree.

Reference: Search Menu

The following image shows the Search menu options that become available after you select Search from the menu bar.



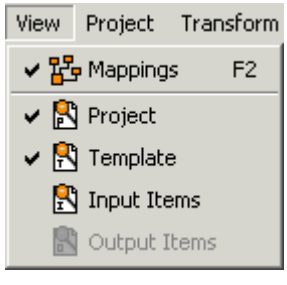
The following table lists and describes the options on the Search menu.

Option	Description
Find	Opens the Find dialog box, enabling you to search for text or to search for and map input and output items.
Find Next	Finds the next instance in the search.

For more information on performing searches in iWay Transformer, see *Using the Find Tool* on page 2-37.

Reference: View Menu

The following image shows the View menu options that become available after you select View from the menu bar.



The following table lists and describes the options on the View menu.

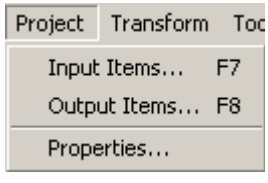
Option	Description
Mappings	Displays the mapping lines between items in the Input Items Structure pane and items in the Output Items Structure pane. The lines represent the data association between the items.
Project	Opens the project pane on the left side of iWay Transformer. This pane displays a hierarchical view of your project and its components.

Option	Description
Template	Adds a View Template tab to the interface. Enables you to view the XML representation or the current mapping for the current project.
Input Items	Adds the View Input Files tab to the interface. Enables you to view the content of structure and transform input files.
Output Items	Adds the View Output Files tab to the interface. Enables you to view the content of structure.

Note: A check mark indicates the view is active. For more details and examples of these views, see *Working With Project Views* on page 2-13.

Reference: Project Menu

The following image shows the Project menu options that become available after you select Project from the menu bar.



The following table lists and describes the options on the Project menu.

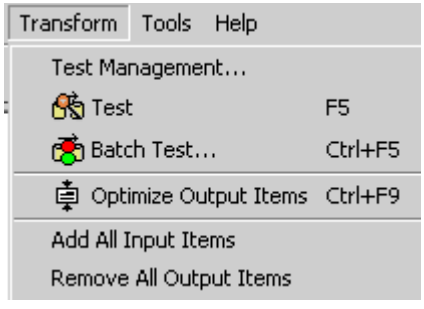
Option	Description
Input Items	Opens the Input Items Properties dialog box, which enables you to specify the properties of input items.
Output Items	Opens the Output Items Properties dialog box, which enables you to specify the properties of output items.
Properties	Opens the Project Properties dialog box, which enables you to set the properties for the project.

For more information on the Input and Output Items Properties dialog boxes, see Chapter 4, *Configuring Input Items* and Chapter 5, *Configuring Output Items*.

For a description of the Project Properties dialog box, see *Adjusting Project Properties* on page 2-25.

Reference: Transform Menu

The following image shows the Transform menu options that become available after you select Transform from the menu bar.

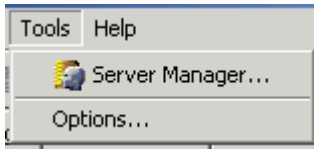


The following table lists and describes the options on the Transform menu.

Option	Description
Test Management	Enables you to manage configurations to run the transform. For more information, see *Adding Test Profiles* on page 2-30.
Test	Runs a test transform without creating output files.
Batch Test	Creates a batch test of the transform.
Optimize Output Items	Removes parent level tags with empty children.
Add All Input Items	Adds all input item nodes to the output structure.
Remove All Output Items	Removes all output item nodes from the output structure.

Reference: Tools Menu

The following image shows the Tools menu options that become available after you select Tools from the menu bar.

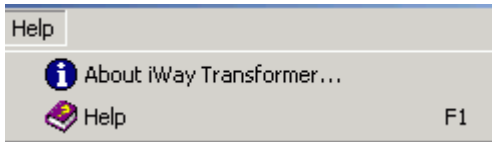


The following table lists and describes the options on the Tools menu.

Option	Description
Server Manager	Opens the Server Manager, which lists the available server configurations. For more information, see *Managing Server Connections* on page 2-34.
Options	Opens the Options dialog box which contains the following tabs: **General.** Options for customizing the interface. For more information, see *Customizing the User Interface* on page 2-20. **Mapping.** Input and output mapping options. **JDBC Drivers**. Enables you to manage the list of JDBC drivers by adding or removing them as required. **Encoding.** Contains encoding options for project and template files.

Reference: Help Menu

The following image shows the Help menu options that become available after you select Help from the menu bar.



The following table lists and describes the options on the Help menu.

Option	Description
About iWay Transformer	Opens the About window, which shows the version of iWay Transformer you are currently using.
Help	Opens help for iWay Transformer.

Reference: iWay Transformer Toolbar Buttons

The following table includes an image of each toolbar button and describes its function.

Button	Function
	Starts a new project.

Button	Function
	Opens a project.
	Saves a project.
	Publishes a project.
	Undoes the last action.
	Redoes the last action.
	Cuts a selected object.
	Copies a selected object.
	Pastes a cut or copied object.
	Moves an item up.
	Moves an item down.
	Opens the iWay Transformer Find tool.
	Shows or hides mappings.
	Performs a batch test.
	Optimizes output items by removing all parent level tags with empty children.
	Opens the Server Manager.

Button	Function
	Opens help for iWay Transformer.

Working With Project Views

This topic describes the views available when working with a project. iWay Transformer contains the following panes in the interface:

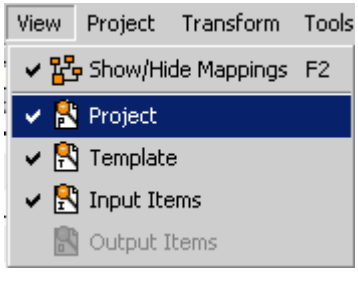
- **Project pane,** on the left, which shows the structure of your project.
- **Template designer pane**, on the right, where you perform most of the work required to prepare a project for transformation. This pane contains the following tabs:
  - Mappings
  - Test Transform
  - View Template
  - View Input Items
  - View Output Items

The Project Pane

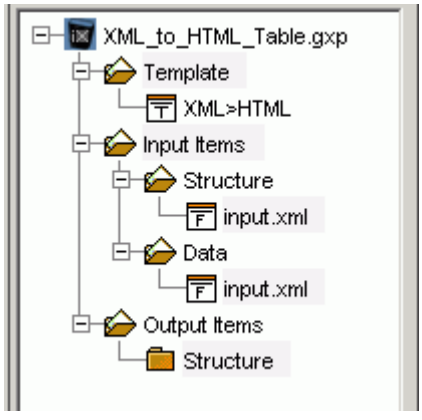
The project pane is the left-most pane of iWay Transformer. You can expand the folders in the pane to view their content. The project pane shows the following information about the current project:

- Project file name.
- Transform template conversion format, for example, XML>CDF.
- Input structure and data files.
- Output structure files.

To display the project pane, you select Project from the View menu as shown in the following image.



The following image shows the contents of a sample project pane.



You can expand or collapse each node to organize the appearance of the project pane.

Mappings Tab

The Mappings tab is the default display for a project and is your work space for building a transform project. You can access the following information from the Mappings tab:

- All or part of the input and output structure tree (expand or collapse the structure nodes).
- The data association between input and output items (represented by lines between items).

  To turn these lines on or off, you click the Show/Hide Mappings button on the toolbar or you select Mappings from the View menu.

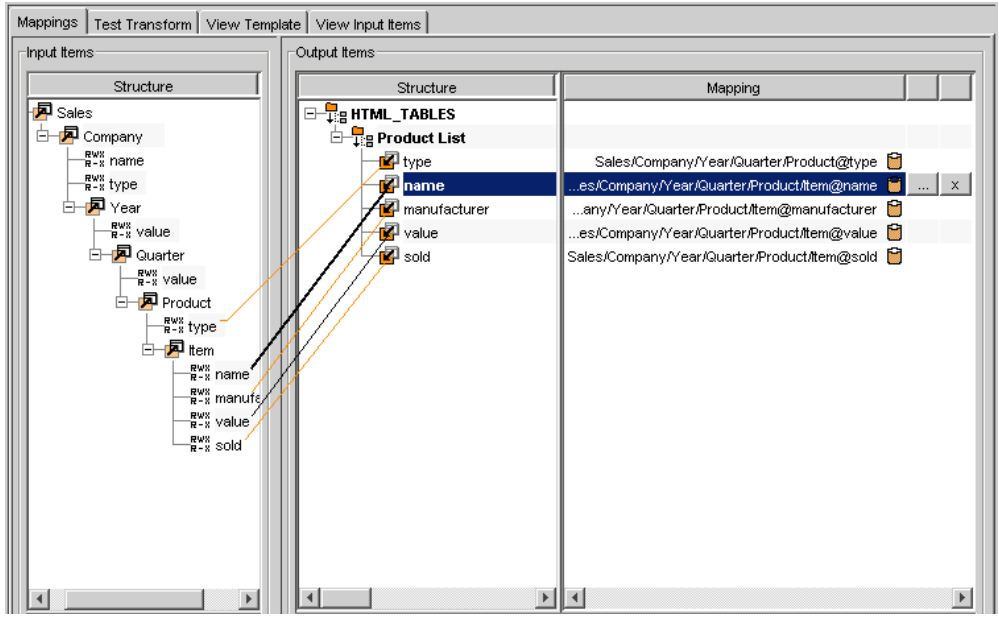
  The colors do not indicate meaning; alternate colors make the lines more distinct.
- Details about a structure item.

  You can obtain details by placing the curser over an input, output, or mapping item.
- The value of an item in the Input Items Structure pane.

  You obtain the value by selecting the item. The value appears below the Input Items Structure pane.
- The mapping value of an item in the Output Items Structure pane.

  You obtain the mapping value by selecting the item. The value appears below the Output Items Structure pane

The following image shows the Mappings tab which includes the panes that show input and output data configurations.



The Mappings tab includes the:

- **Input Items Structure** pane which displays the structure of the input data.
- **Output Items Structure** pane which displays the structure and mapping values of the output data.
- **Mapping** pane which displays the values assigned to the output items.

These panes enable you to work with input and output data structures. When working with these structures, you can:

- Load and configure input or output items.

  You double-click the Structure bar of the Input Items pane to open the Input Items Properties window or the Structure bar of the Output Items pane to open the Output Items Properties window.

  For more information, see Chapter 4, *Configuring Input Items* and Chapter 5, *Configuring Output Items*.

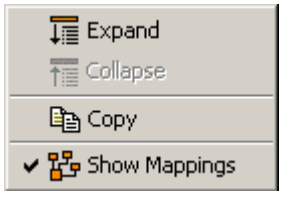
- Drag and drop items from the Input Items Structure pane to the Output Items Structure pane.
- Add or delete a structure item.

The previously described actions and other actions can be performed through the:

- Input Item menu for items in the Input Items Structure pane. For more information, see *Input Items Menu* on page 2-16.
- Output Item menu for items in the Output items Structure pane. For more information, see *Output Items Menu* on page 2-17.
- Value Item menu for items in the Mapping pane. For more information, see *Value Items Menu* on page 2-19.

Reference: Input Items Menu

To view options related to items in the Input Items Structure pane, you right-click an input item. The Input Items menu appears, as shown in the following image.



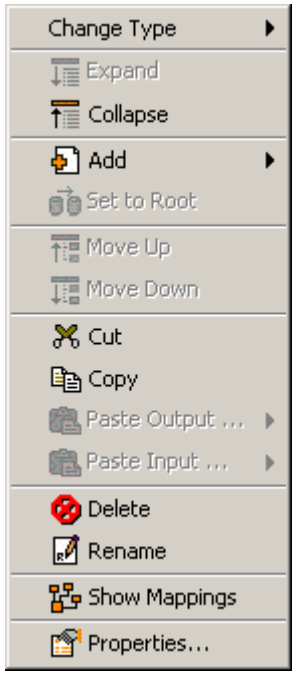
From the Input Items menu, depending on the item you selected, you can expand or collapse the item within the structure tree, copy the item, and show or hide mappings between input and output items.

The following table lists and describes the Input Items menu options.

Option	Description
Expand	Expands the item currently selected in the structure tree.
Collapse	Collapses the item currently selected in the structure tree.
Copy	Copies the selected input item and enables you to paste it in the output.
Show Mappings	Displays the mapping lines between all items in the Input Items Structure pane and the Output Items Structure pane.

Reference: Output Items Menu

To access options related to items in the Output Items Structure pane, you right-click an output item. The Output Items menu appears, as shown in the following image.



The following table lists and describes the options in the Output Items menu.

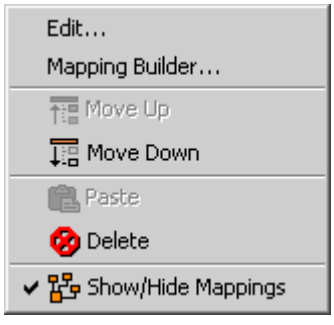
Option	Description
Change Type	Enables you to change the item type to one of the following: • Parent • Element • Attribute • Comment • Content • CDATA: You can define your output item as a CDATA section if it contains many instances of "<" or "&" characters, and you want these characters to be ignored by a parser.
Expand	Expands the selected item.

Option	Description
Collapse	Collapses the selected item.
Add	Enables you to build onto the existing output structure by adding one or more structure items. The item can be one of the following: • Parent • Element • Attribute • Comment • Content • CDATA • iWay XML Request Available only when the output type is iWay XML request. • iWay Embedded XML Request Available only when the output type is iWay embedded XML request.
Set to Root	Uses this element to represent the XML root.
Move Up	Moves the selected item up the output structure tree.
Move Down	Moves the selected item down the output structure tree.
Cut	Cuts the selected output item.
Copy	Copies the selected output item.
Paste Output	Pastes the cut or copied output item.
Paste Input	Pastes the cut or copied input item.
Delete	Removes an item from the Output structure. A confirmation dialog box appears for this option.
Rename	Enables you to rename the selected output structure item.
Show Mappings	Displays mapping lines between the Input and Output Items Structure panes.

Option	Description
Properties	Opens the Output Tag Properties dialog box which shows the properties of the selected structure item. For more information on this dialog box, see Chapter 5, *Configuring Output Items*.

Reference: Value Items Menu

To view options related to an item in the Mapping pane, you right-click the item. The Value Items menu opens as shown in the following image.



The following table lists and describes the options on the Value Items menu.

Option	Description
Edit	Opens the Edit dialog box which enables you to alter the mapping by changing an input item or adding a function, constant, or expression to the mapping.
Mapping Builder	Opens the Mapping Builder which helps you to create mappings for your transform.
Move Up	Moves the selected item up the output structure tree.
Move Down	Moves the selected item down the output structure tree.
Paste	Pastes a copied or cut item.
Delete	Removes the selected mapping.
Show Mappings	Displays the mapping lines between all items in the Input Items Structure pane and the Output Items Structure pane.

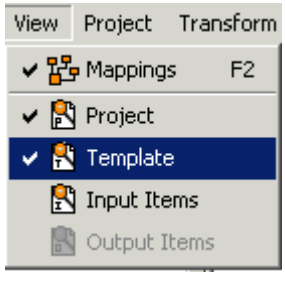
Test Transform Tab

You use the Test Transform tab to test transforms that reflect your changes. When you select this tab, a box opens informing you that the transform is proceeding. After the transform finishes, the output appears. By selecting from the options (represented by icons) that appear above the transform, you can print the transform, save it, refresh the display, or delete the results.

View Template Tab

The View Template tab displays the template file which describes how to transform the input data. The template file reflects changes made in the Mappings tab.

To activate the View Template tab, you select Template from the View menu, as shown in the following image.



View Input Items Tab

The View Input Items tab displays the contents of the input structure file and the input data file. Option buttons enable you to view either a structure file or a data file.

You can print the selected file by clicking the printer icon on the tab. The stop icon next to the printer icon enables you to stop the loading process when loading input files into a project.

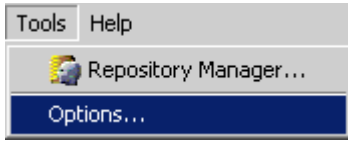
View Output Items Tab

The View Output Items tab displays the output structure. You can print the selected file by clicking the printer icon on the tab. The stop icon next to the printer icon enables you to stop the loading process.

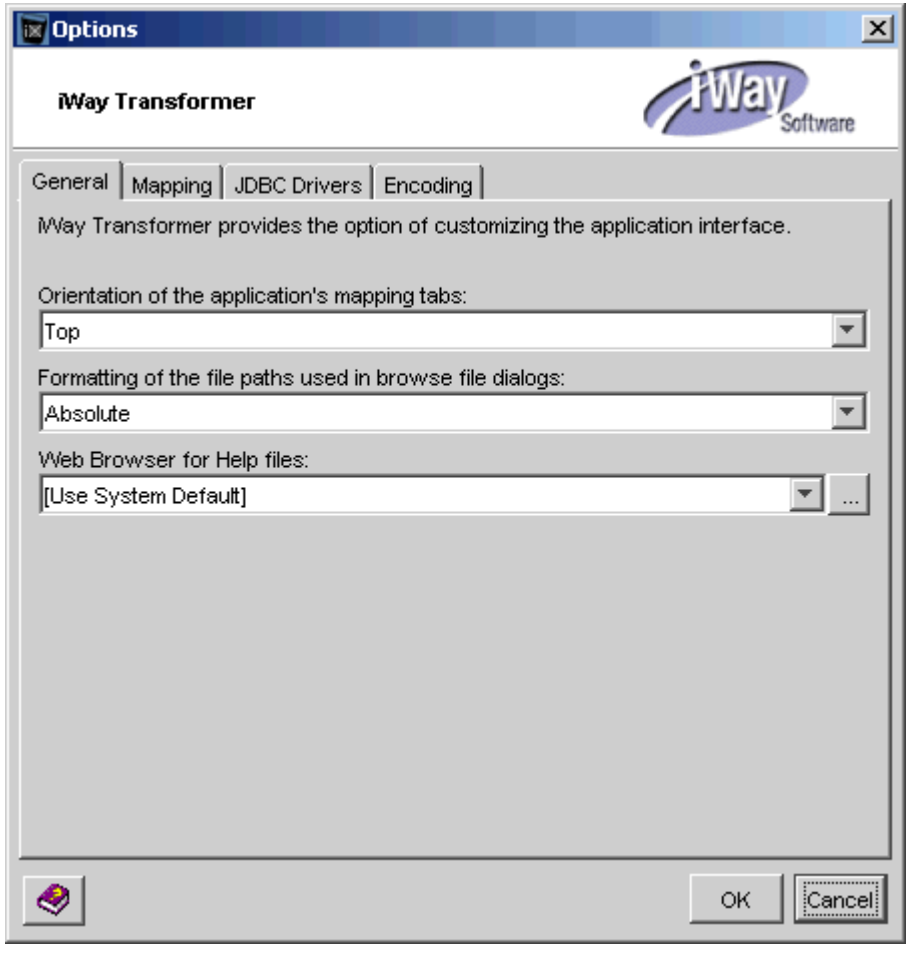
Customizing the User Interface

You can make minor changes to the iWay Transformer user interface according to your preferences. For example, you can choose the location of project tabs and set the file path formatting used in the browse file dialogs.

Interface options are available through the Options dialog box which you access by selecting Options from the Tools menu as shown in the following image.



When the Options dialog box opens, the General tab appears by default as shown in the following image.



General Tab

The General tab includes drop-down lists where you can select options for setting the appearance of the interface.

- Tab Placement (Orientation of the application's mapping tabs)

  Specifies where the Mapping tab appears within the working pane of iWay Transformer. The options are Top, Left, Bottom, or Right.

- Files (Formatting of the file paths used in browse file dialogs)

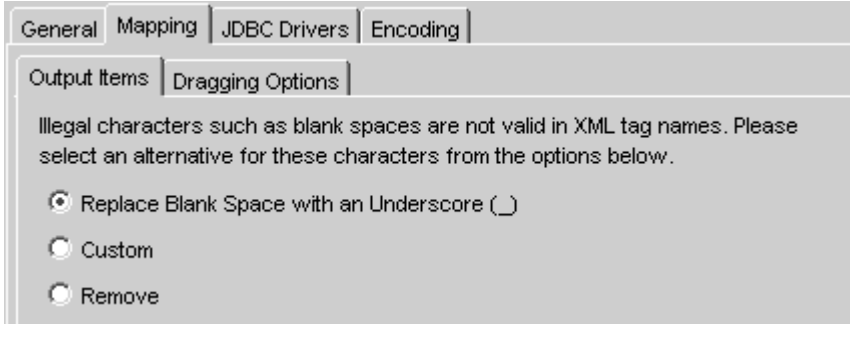
  Enables you to select the format of a file path in iWay Transformer. The options are Absolute or Relative path.

- Help Files (Web Browser for Help files)

  Enables you to select a Web browser to display iWay Transformer online help.

Mapping Tab

The Mapping tab includes two subtabs as shown in the following image. The Output Items tab has option buttons for selecting an alternative to illegal characters. You use the Dragging Options tab to map between input and output items.



The following options are available on the Output Items subtab:

- Replace Blank Space with an Underscore (_)

  You choose this option to replace blank spaces with an underscore.

- Custom

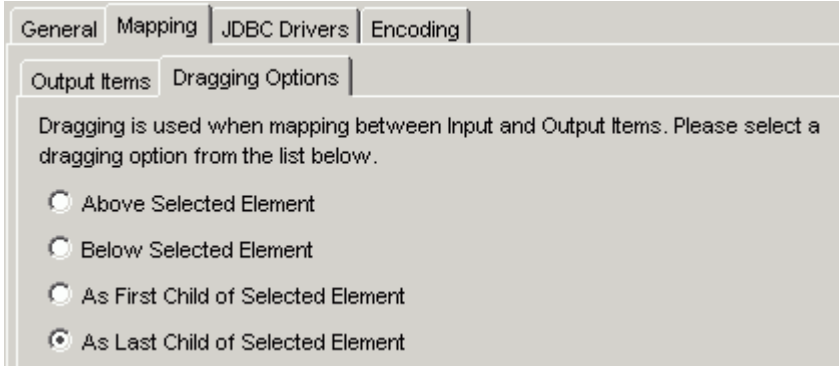
  You choose this option to define a character to use in the place of a blank space.

- Remove

  You choose this option to remove blank spaces.

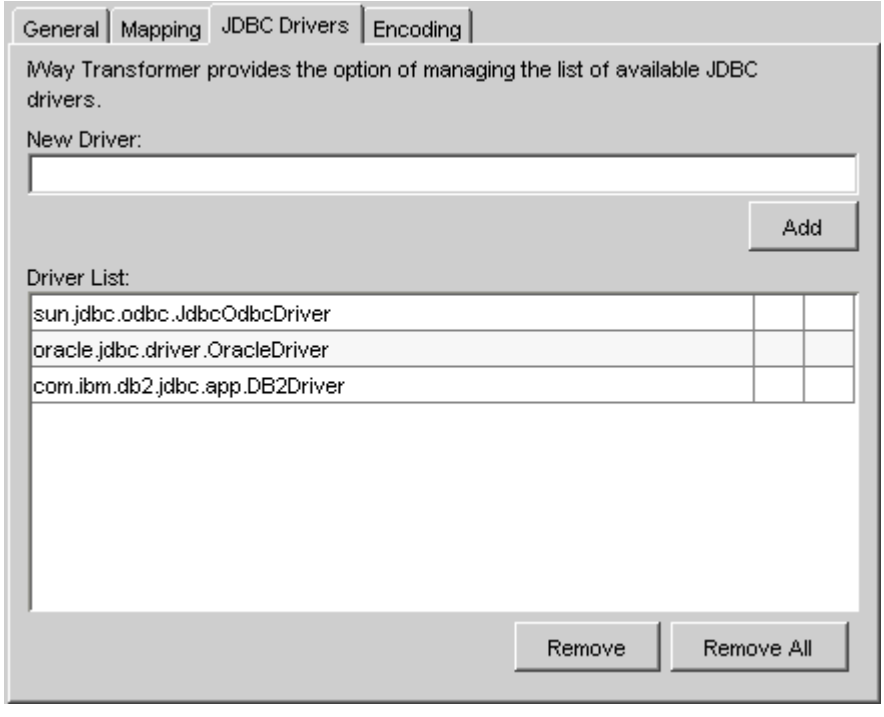
After dragging an item from the Input Items pane to the Output Items pane, you can select whether to have the item placed above or below a selected element or whether to have an item placed as the first or last child of a selected element.

The following image shows the Dragging Options tab with option buttons for the previously described actions.



JDBC Drivers Tab

You can use JDBC, with the JDBC replace function, to replace output values in iWay Transformer. The Sun, Oracle, and IBM drivers are shown as options in the JDBC Drivers tab as shown in the following image.



iWay Transformer ships with the Sun JDBC driver (as part of Java). The other two drivers (Oracle and IBM) do not ship with iWay Transformer. They are listed because they are common JDBC drivers.

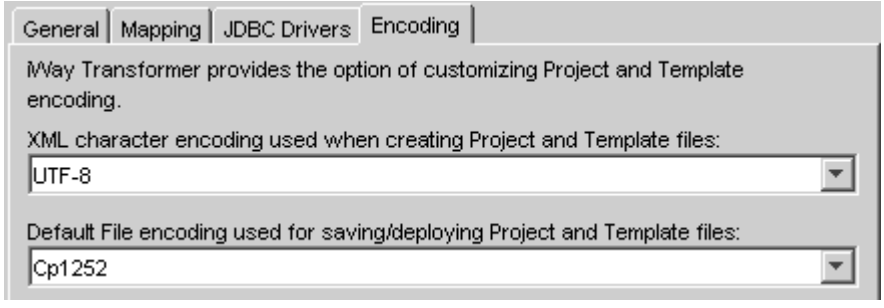
To use one of the drivers, you must copy the actual driver to the following directory:

```
C:\Program Files\iWay55sm\tools\transformer\jdbc_drivers
```

Then, you must add the file name to the list of JDBC drivers by specifying the driver in the New Driver field and click the *Add* button.

Encoding Tab

The following image shows the Encoding tab which provides the option of customizing Project and Template encoding.



The following two encoding options are available from the drop-down lists:

- **Character encoding.** You can select the encoding to use for the characters in the file.
- **File encoding.** You can select the default file encoding. You choose this if you are saving or deploying project and template files to a machine that uses a different language format.

Adjusting Project Properties

The Project Properties dialog box enables you to view and adjust parameters and properties defined for a project such as project creation date and custom functions.

The Project Properties dialog box includes the following tabs:

- The **General** tab enables you to provide template metadata. This is optional and does not affect the transform. The values you type are stored in your template file as template metadata and are not added to your output.
- The **@REPLACE Function** tab provides a way to match and replace input data values through replace functions. Replace functions work like custom functions in that you must first define the function and then, apply it in the output item mapping value definition that you want to affect. For more information, see Chapter 6, *About Functions*.
- The **Custom Functions** tab enables you to write a custom function when an iWay Transformer pre-defined function does not exist to perform the task you require. Custom functions must be written in Java and stored in the following directory to be available for use with iWay Transformer during design time:

  `iWay55sm\tools\transformer\custom_functions`

  Custom functions must be published for use at run time.

For more information about using functions, see Chapter 6, *About Functions*.

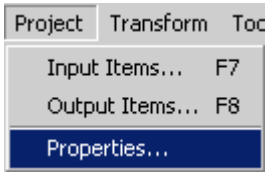
- The **Global Constants** tab enables you to create global constants to use in your output item values. You can use a defined constant in the output items mapping value through the @GETCONSTANT function.

  In the Name column, type the name of the constant you want to create. In the Value column, type its initial value. Note that the value of any global constant may be changed when you run the actual transform from iWay Service Manager.

- The **JDBC Data Source** tab enables you to configure a JDBC connection. You can define multiple JDBC connections or add a connection from an existing project.

- The **XML Namespaces** tab enables you to load XML namespaces from other projects or create your own. For more information on the XML Namespaces tab, see *Configuring XML Namespaces* on page 2-26.

To access the Properties dialog box, you select the Properties option from the Project menu, as shown in the following image.



Configuring XML Namespaces

XML namespaces provide a way to distinguish between duplicate element type and attribute names. A duplication may occur, for example, in an XSLT style sheet or in a document that contains element types and attributes from two different DTDs. An XML namespace is a collection of element type and attribute names. In an XML namespace, an element type or attribute name is uniquely identified by a two-part name: the name of its XML namespace and its local name.

The following example uses XML namespaces to distinguish between two different element types named Address.

```
<Department>
<Name>DVS1</Name>
<addr:Address xmlns:addr="http://www.tu-darmstadt.de/ito/addresses">
<addr:Street>Wilhelminenstr. 7</addr:Street> <addr:City>Darmstadt</
addr:City>
<addr:State>Hessen</addr:State>
<addr:Country>Germany</addr:Country>
<addr:PostalCode>D-64285</addr:PostalCode>
</addr:Address >
<serv:Server xmlns:serv="http://www.tu-darmstadt.de/ito/servers">
<serv:Name>OurWebServer</serv:Name> <serv:Address>123.45.67.8</
serv:Address>
</serv:Server>
</Department>
```

The first Address element type name belongs to the http://www.tu-darmstadt.de/ito/addresses XML namespace. It has a universal (two-part) name of "http://www.tu-darmstadt.de/ito/addresses" and "Address". The second Address element type name belongs to the http://www.tu-darmstadt.de/ito/servers XML namespace and has a universal name of {http://www.tu-darmstadt.de/ito/servers}Address. Thus, each universal name is unique, meeting the requirement that each element type in an XML document have a unique name.

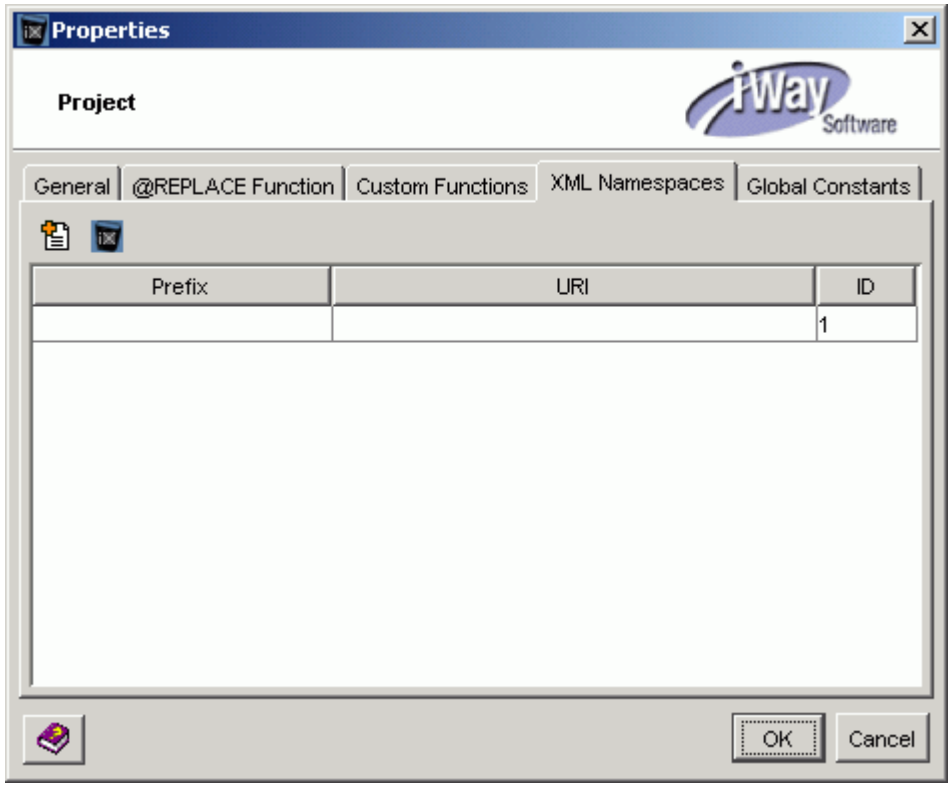
You can configure your project to use XML namespaces by loading in a set of namespaces from an existing project or by creating your own. Either way, ensure you check the Contains Namespace option when configuring your input.

Procedure: How to Load an XML Namespace From Another Project

To load an XML namespace from another project:

1. In the Project Properties dialog box, click the *XML Namespaces* tab.

The following image shows the XML Namespaces tab with areas for prefix, URI, and ID.



2. Click the *Load From Project* button located at the left above the Prefix column.

   The Open dialog box appears.

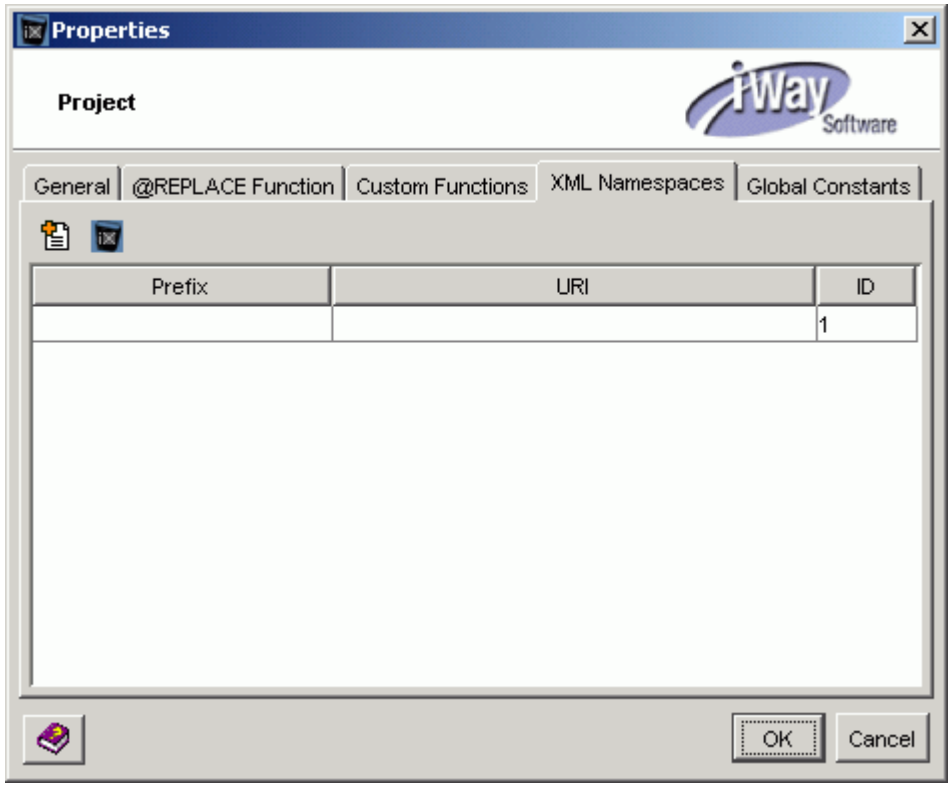
3. Find and select the project whose namespaces you wish to load and click *Open*.

Procedure: How to Create an XML Namespace

To create your own XML namespace:

1. In the Project Properties dialog box, click the *XML Namespaces* tab.

The following image shows the XML Namespaces tab with areas for prefix, URI, and ID.



a. In the Prefix field, type the prefix to be associated with the XML namespace.

b. In the URI field, type the URI associated with the prefix and use the URI to represent the style sheet, reference, or value for the XML namespace.

An identification number for each XML namespace appears automatically in the ID field.

For example, a prefix value of addr with the corresponding URI

```
http://www.tudarmstadt.de/ito/addresses
```

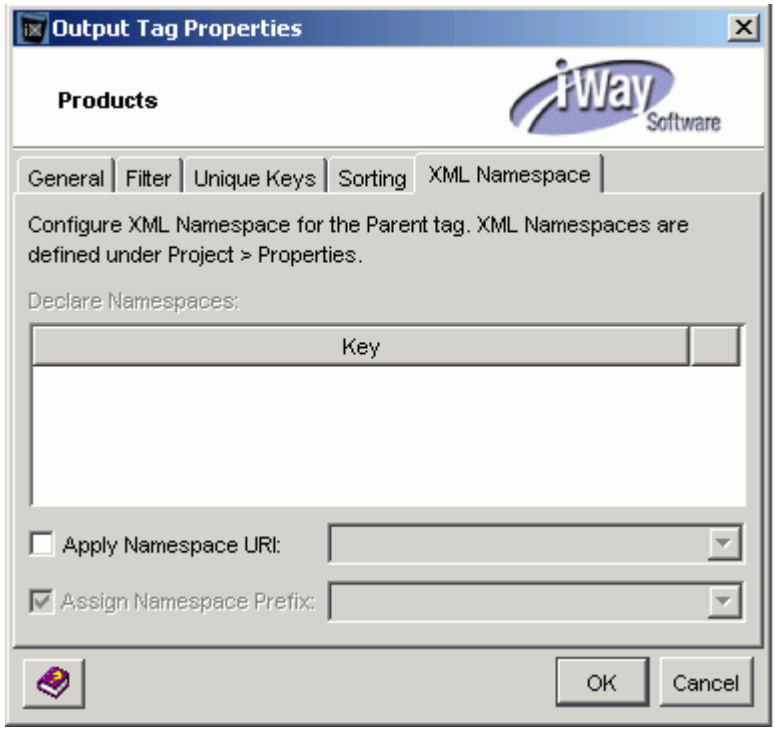
is the equivalent of including the following in your actual data:

```
xmlns:addr="http://www.tu-darmstadt.de/ito/addresses"
```

2. If you want to include more than one XML namespace for your project, click the *Add Row* button to add another row.

3. Click *OK*.

Note: To attach XML namespaces, left-click parents, elements, or attributes in the Output Items Mappings pane. Select *Properties*. When the Output Tag Properties dialog box opens, click the XML Namespaces tab (if it is not already active) and then, select the *Apply Namespace* URI checkbox as shown in the following image.

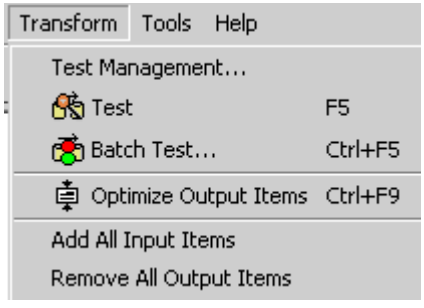


Adding Test Profiles

After creating a transformation project and mapping input and output structures, you must configure a test to preview and execute the transformation. Although iWay Transformer is installed and used on one machine, you can execute your transformations on a remote machine where iWay Service Manager resides.

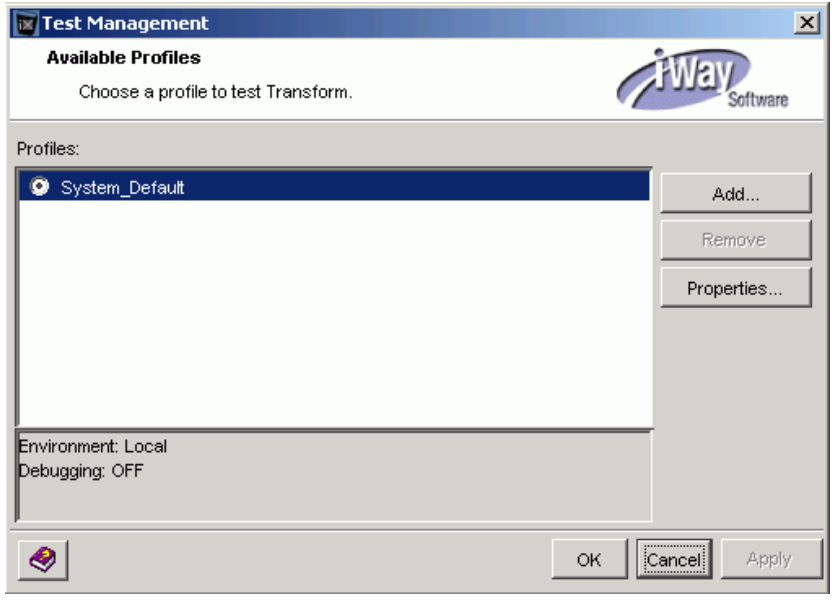
Procedure: How to Add a Test Profile

To add a test profile:



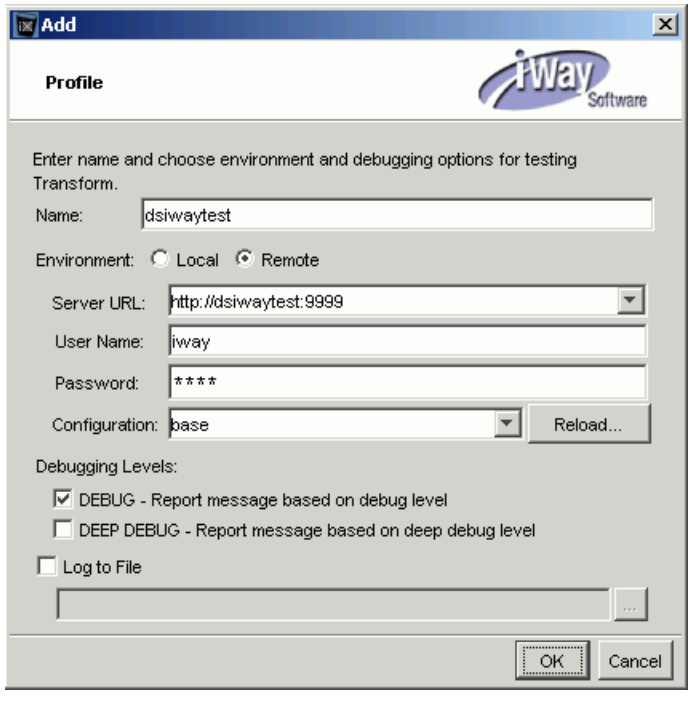
1. From the Transform menu, select *Test Management*.

   The Test Management dialog box opens as shown in the following image.



2. Click *Add*.

The Add dialog box opens where you provide the properties for the test configuration profile as shown in the following image.



a. In the Name field, type a name to identify your test configuration.

b. In the Server URL field, type the URL of the machine where iWay 5.5 SM is installed and iWay Service Manager is available.

c. In the User Name field, type a valid user name for iWay Service Manager.

d. In the Password field, type a valid password for the specified user ID.

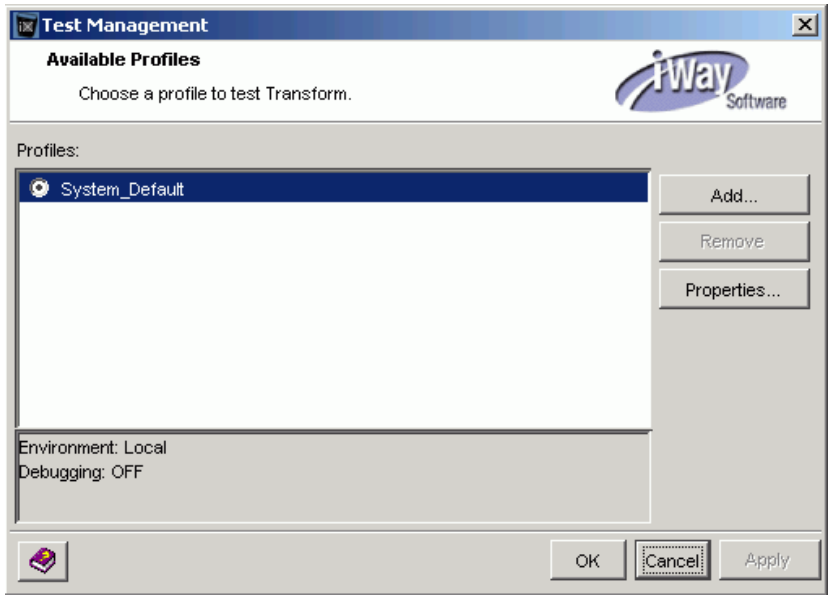
e. In the Configuration field, type or select a configuration from the drop-down list. Click *Reload* to refresh the list of available configurations.

f. From the Debugging Levels option list, select the level of debugging you want to use.

g. To specify a file path for the log file, select *Log to File*.

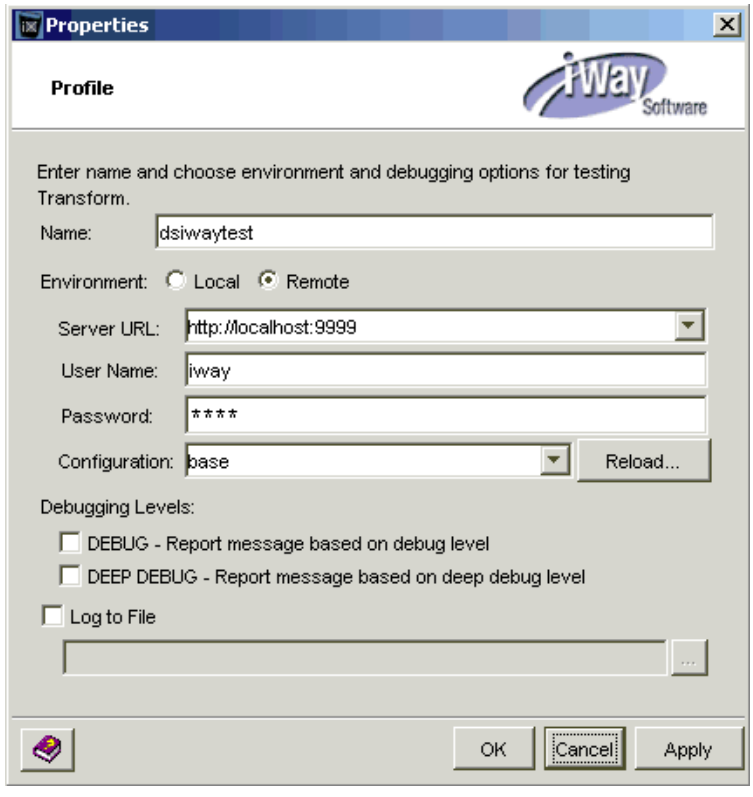
3. Click *OK*.

You are returned to the Test Management dialog box where the test configuration you created is added to the Profiles pane.



4. To modify a test configuration, select a configuration from the list and click *Properties*.

The Properties dialog box opens as shown in the following image.



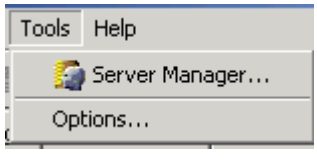
5. Make changes to your configuration properties, if required.
6. Click *Apply*, and then *OK* when you are finished.

Managing Server Connections

The Server Manager lists available servers where you can publish your transformation projects. The Server Manager enables you to add a server connection, configure a server connection, or remove a server.

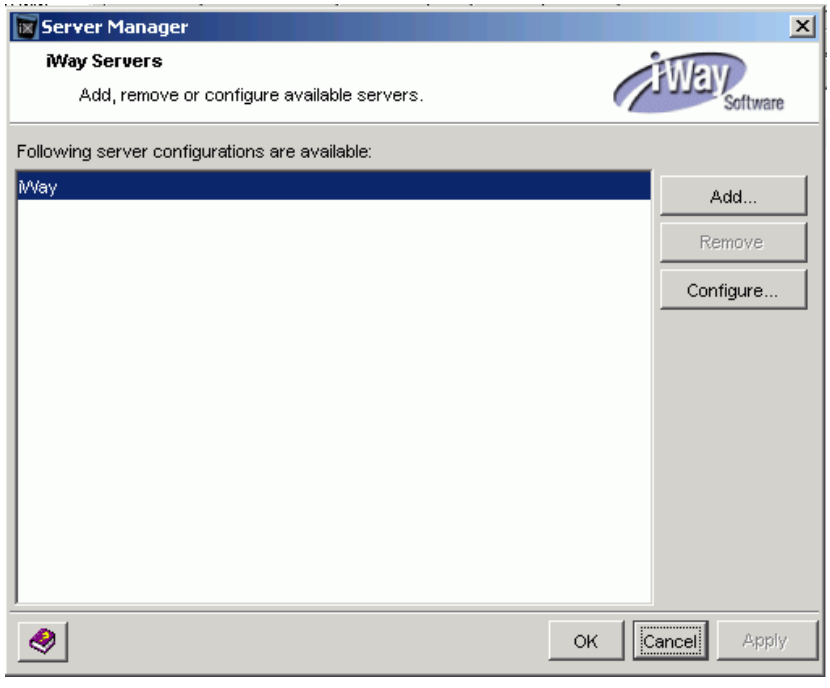
Procedure: How to Add a Server Connection

The following image shows the Tools menu.



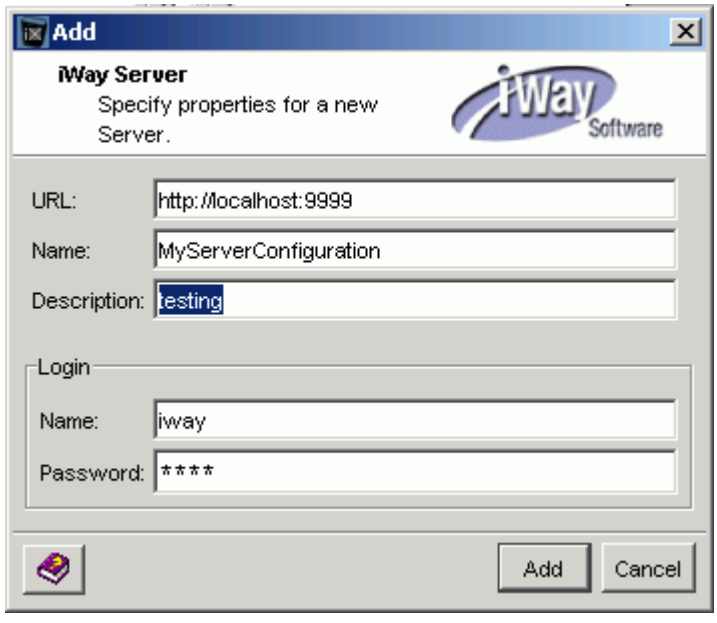
1. From the Tools menu, select *Server Manager*.

   The Server Manager window opens as shown in the following image.



2. In the Server Manager, click *Add*.

The Add dialog box opens as shown in the following image.



3. In the URL field, type the location of the repository you want to add.

   The format of the URL is

   `http://host:port`

   where:

   host

   Is the host machine on which iWay Service Manager is installed. The default value is localhost.

   port

   Is the port on which iWay Service Manager is listening. The default port is 9999.

4. In the Name field, type the name you want iWay Transformer to display.
5. In the Description field, type a description of the server.
6. In the Login pane, type the user ID and password for the iWay Server.
7. Click *Add*.

   The new server connection appears in the list of servers in Server Manager.

Procedure: How to Modify an Existing Server Connection

To modify an existing server connection:

1. In Server Manager, select the server you want to configure, and click *Configure*.

   The Configure dialog box appears, listing the information you typed when you first created the server connection. You can also review the Transforms published on the server.

2. Make the required changes to the server information such as URL, name, or description.
3. After you make your changes, click *OK*.

Procedure: How to Delete a Server Connection

To delete a server connection:

1. In Server Manager, select the server you want to delete and click *Remove*.

   A dialog box appears, asking you to confirm the deletion.

2. Click *Yes*.

Using the Find Tool

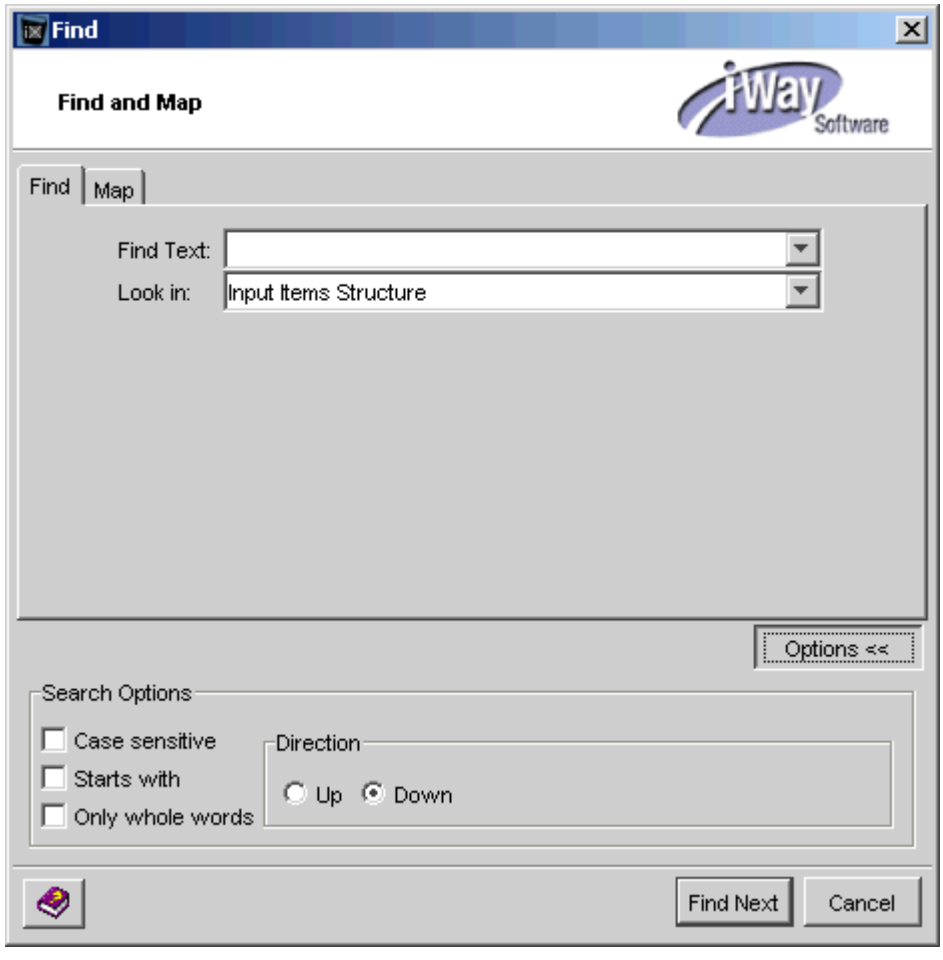
The iWay Transformer Find tool enables you to locate a specific parent, element, or attribute in the input and output structure of a project. You can also find and map a specific parent, element, or attribute within an input structure or an output structure.

Procedure: How to Use the Find Tool

To use the Find tool:

1. From the Search menu, select *Find* or click the *Find* button in the toolbar.

The Find dialog box opens. The Find tab is active by default as shown in the following image.



a. In the Find Text field, type the text you are looking for.

b. From the Look in drop-down list, select the project component you want to search: Input Items Structure, Output Items Structure, View Input Files, View Template, or Test Transform.

c. To expand the Find dialog box and view more options to refine your search, click the *Options* button.

The additional options are:

Case sensitive which retrieves values with the exact case you specify in Find Text.

Starts with which retrieves all items that begin with your entry in Find Text.

Only whole words which retrieves tags or attributes with full names that match your entry in Find Text. For example, if you type "range" in the Find Text field and select the Only whole words option, the search does not retrieve an element named "orange."

Direction which enables you to specify whether to search up or down the file.

2. To find the next occurrence of the defined item, click *Find Next*.

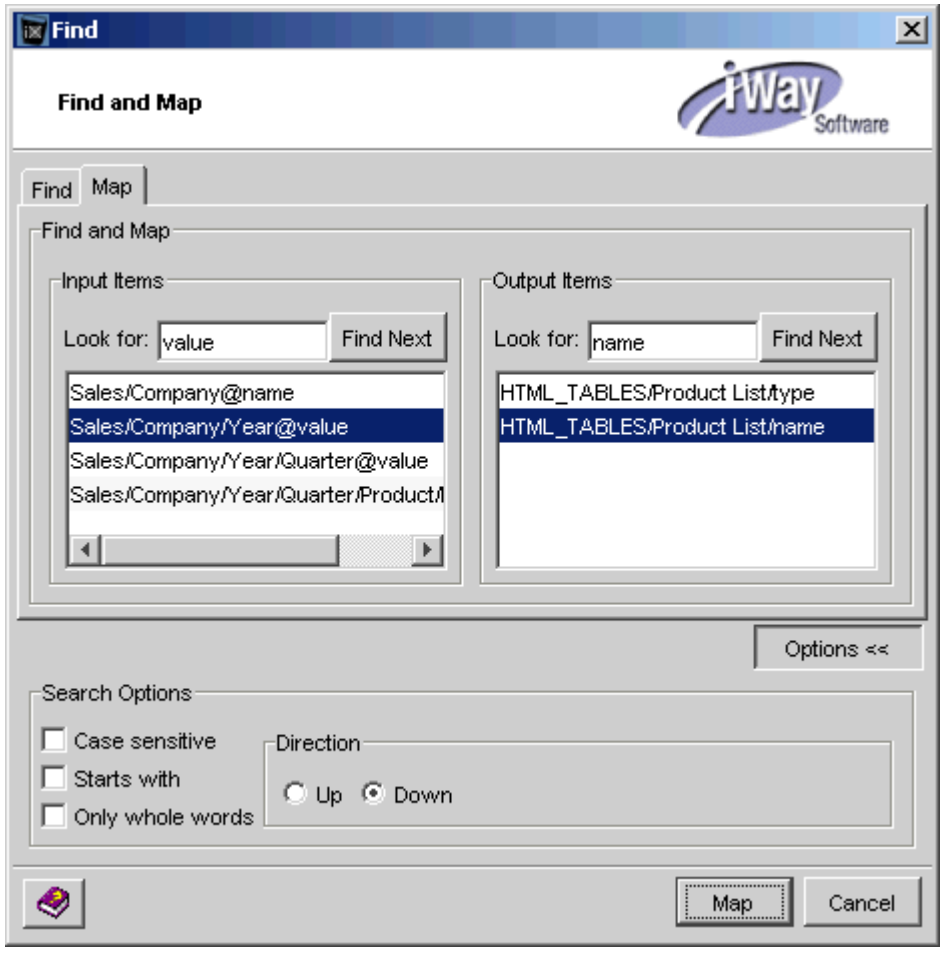
   If the search tool does not find a match for your search, a message appears indicating that your entry was not found.

Procedure: How to Use the Find and Map Option

To find and map a specific parent, element, or attribute within an input structure or an output structure:

1. From the Search menu, select *Find* or click the *Find* button in the toolbar.

The Find dialog box opens as shown in the following image.



a. Click the *Map* tab.

b. Type a value to find and map in the Input Items section.

c. Click *Find Next*.

The path of the value in the input item is mapped in the results pane.

d. Type a value to find and map in the Output Items section.

e. Click *Find Next*.

The path of the value in the output item is mapped in the results pane.

2. To remove a specific path, right-click a path in the results pane and select *Remove Selected Path*. To remove all paths in the results pane, right-click within the results pane and select *Remove All*.

3. If you want to map the selected items, click *Map*.

Customizing Java Properties

You can customize Java properties by modifying the transformer.xml file, which is located in the transformer/bin directory. This file provides a set of properties for advanced options when running iWay Transformer in design time. Modifying these properties is useful for specifying additional Java Virtual Machine (JVM) options, setting Java system properties, and registering Java class and JAR files in the classpath.

Configuring JVM settings can improve the performance of iWay Transformer or resolve errors. The most common settings include the size of the Java heap and stack, which determine memory availability for Java programs and the JVM. If sufficient memory is not available, errors can occur. The heap size affects performance, as it determines how often garbage collection occurs.

If you encounter performance problems or receive out of memory exception errors, adjusting the Java heap and stack sizes is a good option. The following are the most common JVM memory settings:

- -Xss###M

  Sets the Java thread stack size.

- -Xmx###M

  Sets the maximum Java heap size.

- -Xms###M

  Sets the initial Java heap size.

The following is an example of the transformer.xml file. Note the JVM_OPTION parameter, which can be used to increase memory allocation while running a JVM with an extended heap size:

```
<APPCONFIG version="5.5">
   <Application name="transformer" class="">
      <Properties>
          <Property name="POST_CLASSPATH" description="external classes
          to be appended to classpath" value="" />
          <Property name="PRE_CLASSPATH" description="external classes
          to be prepended to class path" value="" />
          <Property name="JVM_OPTION" description="options to be applied
          at jvm startup" value="-mx512m" />
          <Property name="JAVA_LIB_PATH" description="dlls to be added to
          java.library.path" value="" />
      </Properties>
   </Application>
</APPCONFIG>
```

Note: For any changes made in the iwaytransformer.xml file to take effect, you must first close and restart iWay Transformer.

On Windows, the Register Libraries function enables you to add Java classes and .JAR files to the classpath. You can use this function to add third-party drivers, for example, JDBC. You can also specify additional library directories that may be required when the third-party Java classes require dynamic link libraries or shared objects (depending on the platform in use).

To add an entry to the beginning of the classpath, use the PRE_CLASSPATH property. To add an entry to the end of the classpath, use POST_CLASSPATH.

CHAPTER 3

Working with Projects

Topics:

- Creating Projects
- Saving Projects
- Opening Existing Projects
- Opening Sample Projects
- Testing Transformations
- Publishing Projects
- Using Templates
- Viewing Templates

An iWay Transformer project contains all the information associated with a particular transformation, that is, a project file containing the transformation mappings and the input and output structure and data files.

A project is an entire configuration set for a transformation. It contains the template file and all other relevant configuration information and settings, such as input and output data structure and format. Project files have .gxp extensions.

This section describes how to design and work with projects, whether new or existing, and how to perform a transformation.

Creating Projects

You can create a new project using the Project Wizard or manually.

Creating a New Project

The Project Wizard provides a quick way to create a simple transformation project. It guides you through four major steps to build the project. After you create the project, you can make changes using the editing and configuration tools of iWay Transformer. The following procedure shows you how to create a project with the Project Wizard.

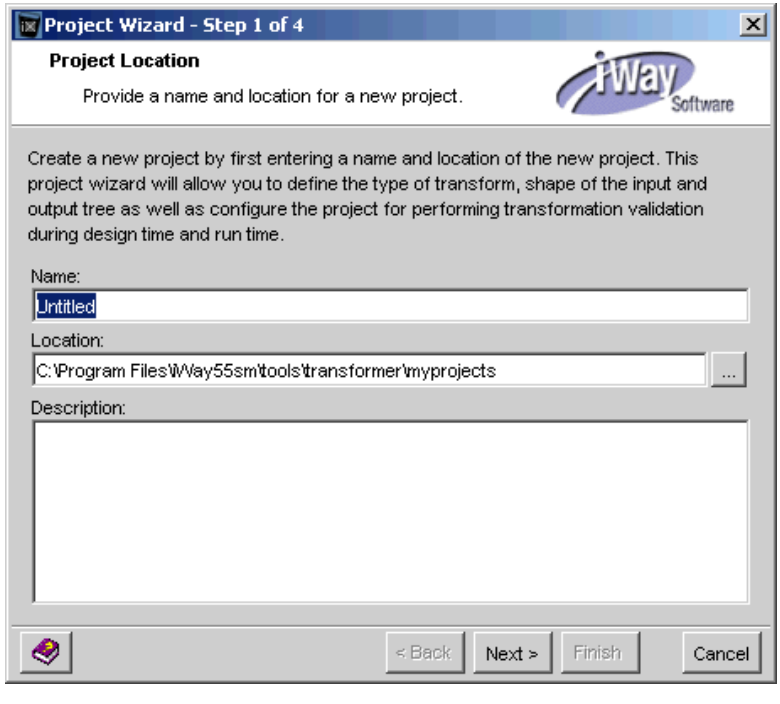
When creating a project manually, you specify the same aspects of a project as with the Project Wizard, but you use the project pane and its associated dialog boxes. For information on how to create a project manually, see *How to Create a New Project Manually* on page 3-7.

Procedure: How to Create a Project With the Project Wizard

To create a project using the Project Wizard:

1. From the File menu, select *Project Wizard*.

   The Project Wizard - Project Location dialog box opens as shown in the following image.



a. In the Name field, type a name for your new project.

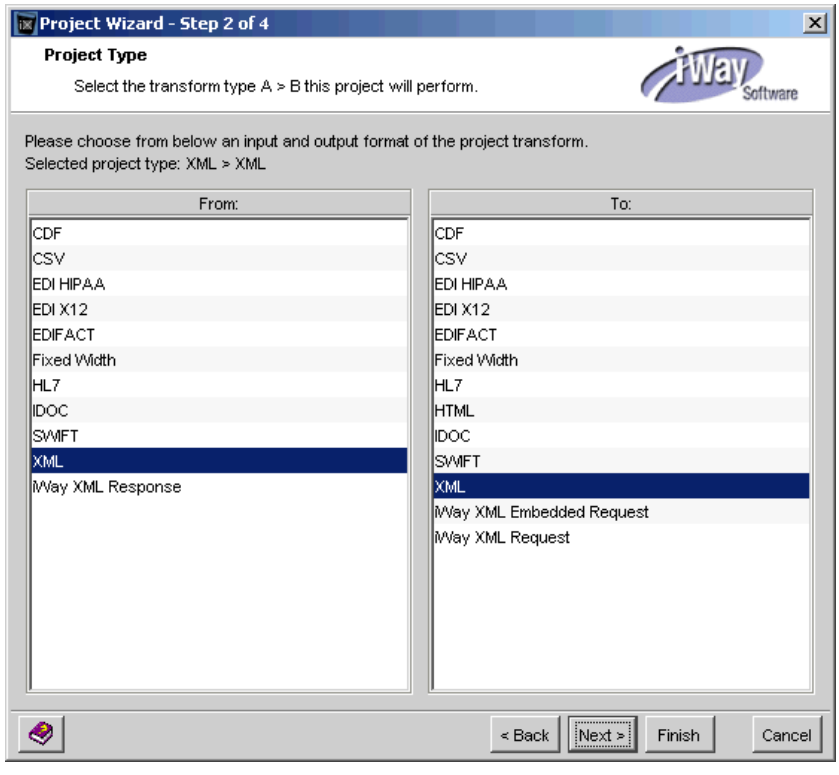
Note: When naming a project, iWay Transformer adds a file extension to the name for you.

b. In the Location field, type or choose a path for the project working directory.

c. In the Description field, type a project description.

2. Click *Next*.

The Project Wizard - Project Type dialog box opens as shown in the following image.

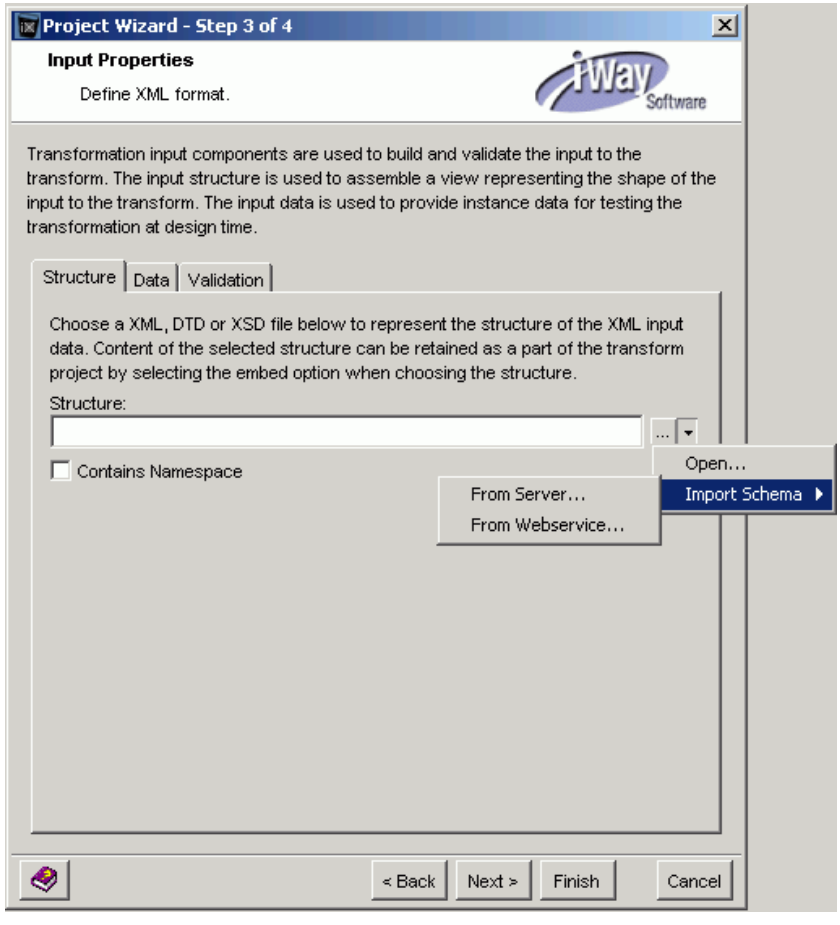


a. From the list in the From pane, select the format of your input data.

b. From the list in the To pane, select the format of your output.

3. Click *Next*.

The Project Wizard - Input Properties dialog box opens with the Structure tab active as shown in the following image.



a. In the Structure field, type the name of a DTD, XSD, or XML file to represent the structure of the input data or click the *Browse* button to select a file.

You can also click the arrow to the right of the Structure field, select *Import Schema* and click *From Server* or *From Webservice*.

Note: No external Web services are supported, only iWay Web services, which are also known as iWay Business Services.

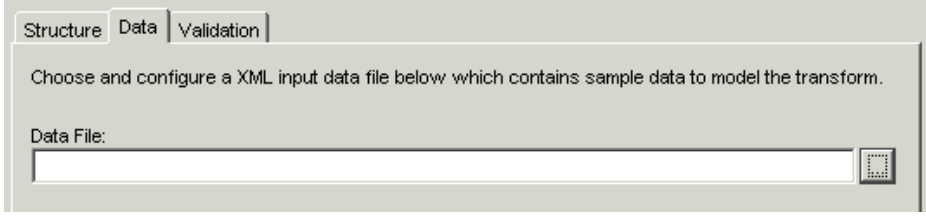
b. Select the *Contains Namespaces* option if the DTD or XSD contains a valid namespace.

The Data and Validation tabs are activated when available for the selected Input structure.

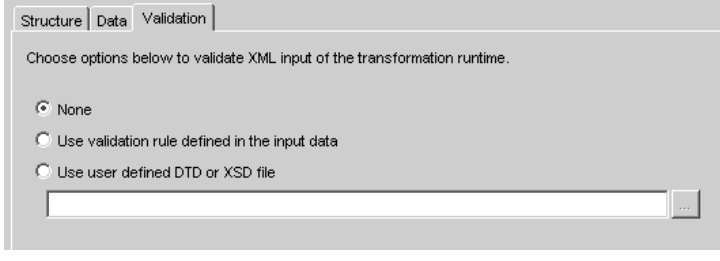
c. Complete the fields on these tabs if desired.

The Data tab enables you to select an XML input data file to configure, which contains sample data to model the transformation. You must supply an input data file for testing transformations.

The following image shows the Data tab with an empty Data File field.

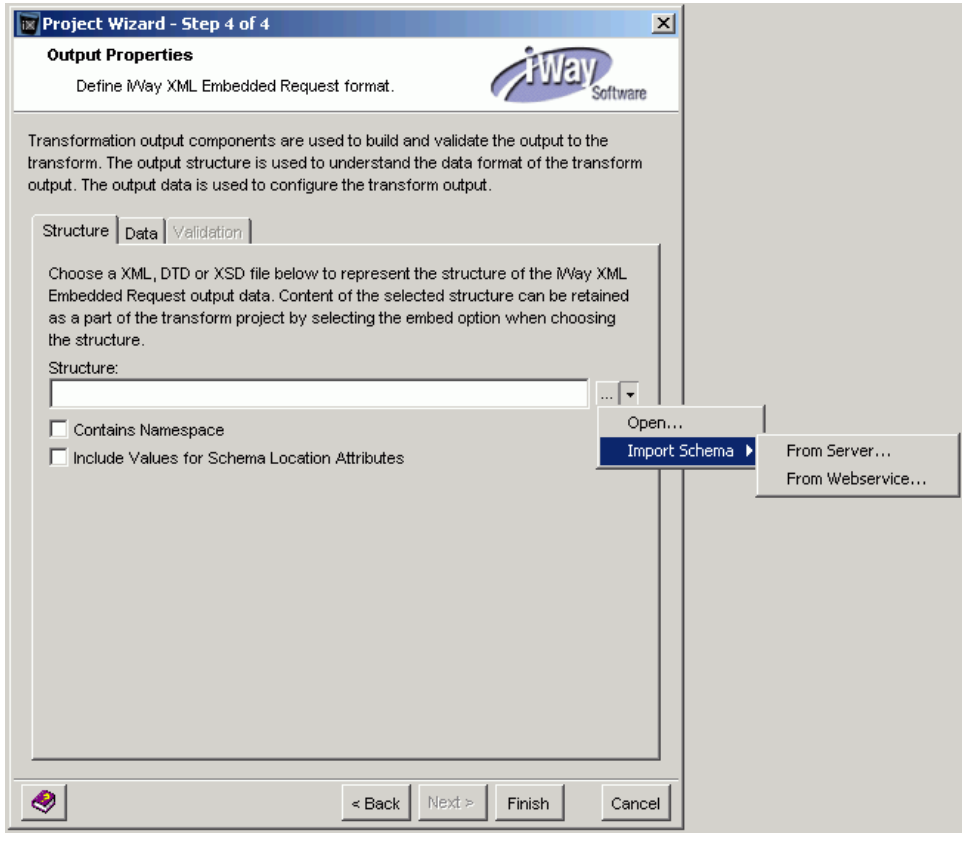


The Validation tab enables you to select validation options for the XML input during run time as shown in the following image.



4. Click *Next*.

The Project Wizard - Output Properties dialog box opens.



a. In the Structure field, type the name of a file to represent the structure of the output data or click the *Browse* button to select a file.

You can also click the arrow to the right of the Structure field, select *Import Schema* and click *From Server* or *From Webservice*.

Note: No external Web services are supported, only iWay Web services, which are also known as iWay Business Services.

b. Select the Contains Namespaces and Includes Values for Schema Location Attributes options if required.

5. Click *Finish*.

The Project Wizard closes.

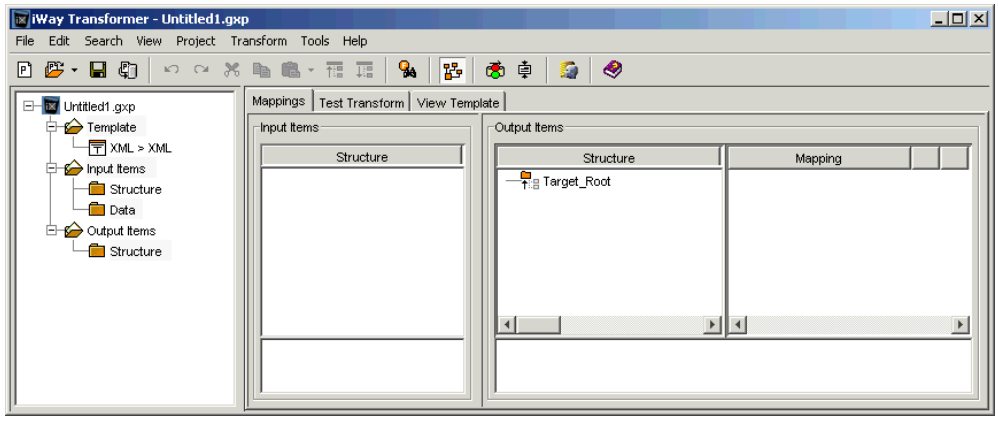
Your new project now exists in the directory you named during the creation process and is accessible from iWay Transformer.

Procedure: How to Create a New Project Manually

To create a new project manually:

1. Click *New Project* on the toolbar or select *New* from the File menu.

   iWay Transformer opens the project pane to display the hierarchy of a project named Untitled1.gxp, as shown in the following image. The Structure pane for Output Items shows a Target_Root Structure.



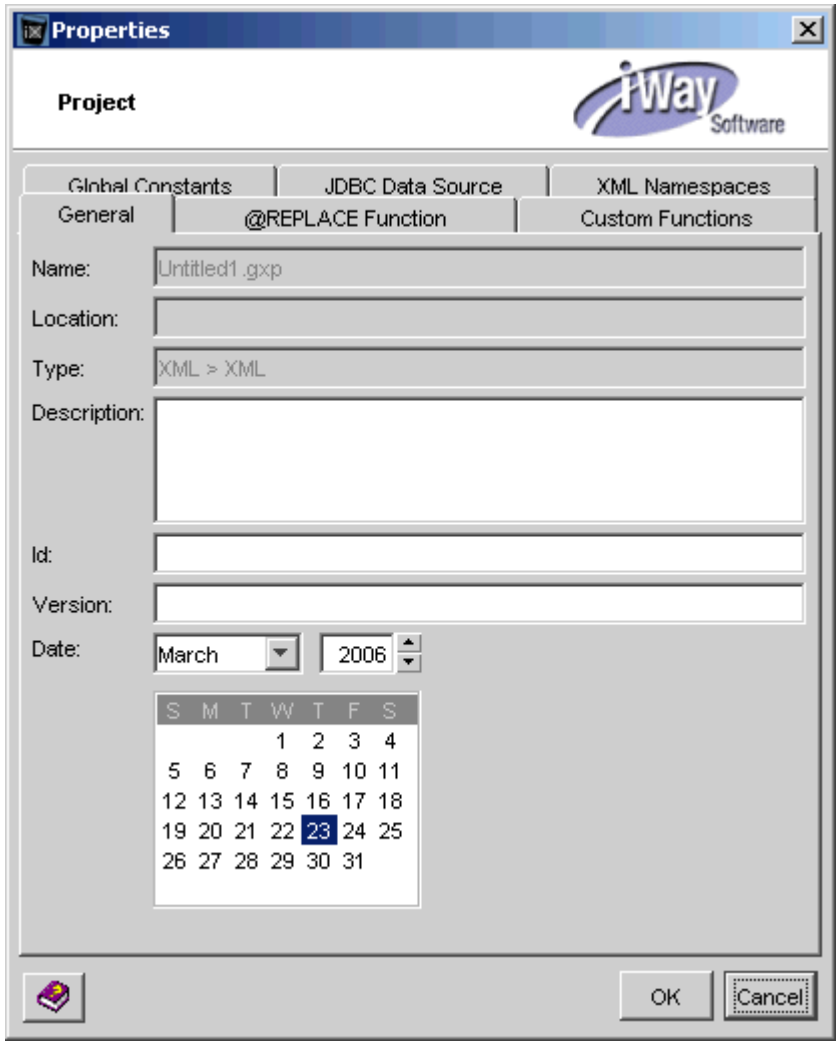
   The default template used by a project is XML>XML. You can change this template.

2. Save the project.

   iWay Transformer now displays the project name on the project pane.

3. From the Project menu, select *Properties*.

The Project Properties dialog box opens with the General tab active as shown in the following image. This tab displays project name, location, and the date. This information is optional and does not affect the transform. The values you type are stored in your template file as template metadata and are not added to your output.



a. In the Description field, type a description of the project.

b. In the Id field, type the project identification.

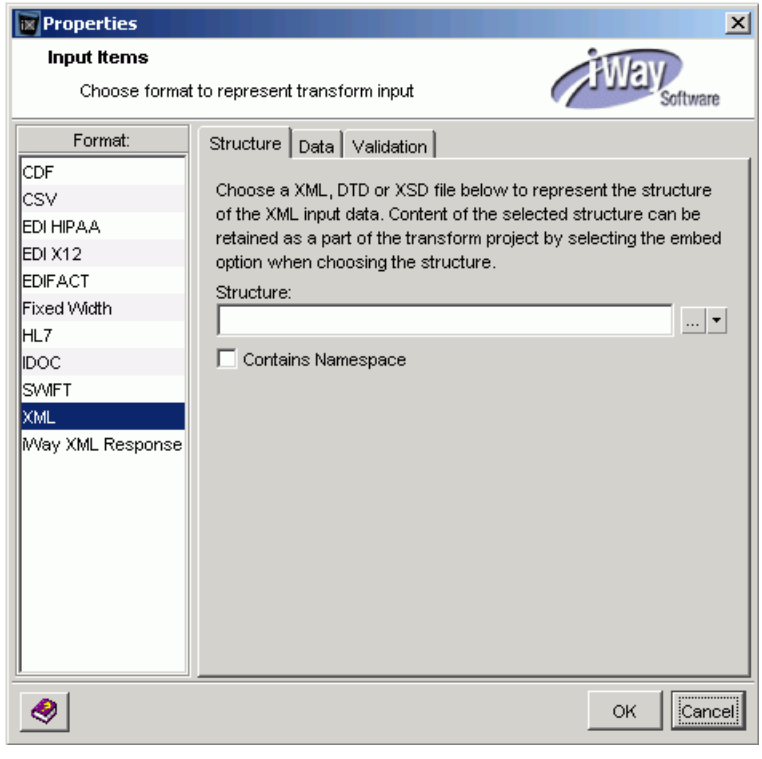
c. In the Version field, type the version number.

4. Click *OK*.

The Project Properties dialog box closes.

5. To load the input data format and structure information, from the Project menu, select *Input Items* or right-click the *Input Items Structure* bar and select *Input Items Properties*.

   The Input Items Properties dialog box opens as shown in the following image.



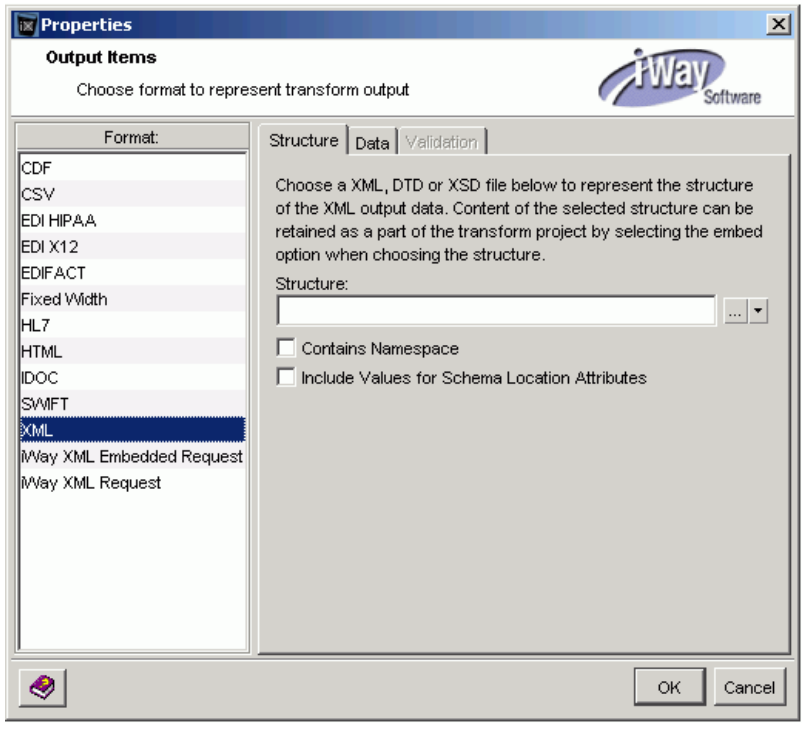
   a. From the Format list, select the input type.

   b. Make the appropriate selections in the Structure, Data, and Validation tabs. For a description of these options, see Chapter 4, *Configuring Input Items*.

6. To load, click *OK*.

7. To load the output data format and structure information, from the Project menu, select *Output Items* or right-click the *Output Items Structure* bar and select *Output Items Properties*.

The Output Items Properties dialog box opens as shown in the following image.



a. From the Format list, select the output type.

b. Make the appropriate selections in the Structure, Data, and Validation tabs. For a description of these options, see Chapter 5, *Configuring Output Items*.

8. To load, click *OK*.

You cannot pre-load an output structure for an HTML output data format, therefore, you must build your output structure using the structure components provided in iWay Transformer.

An output item value can be built from an input item, a function, a constant value, or an expression. For details about assigning output values, see Chapter 5, *Configuring Output Items*.

Saving Projects

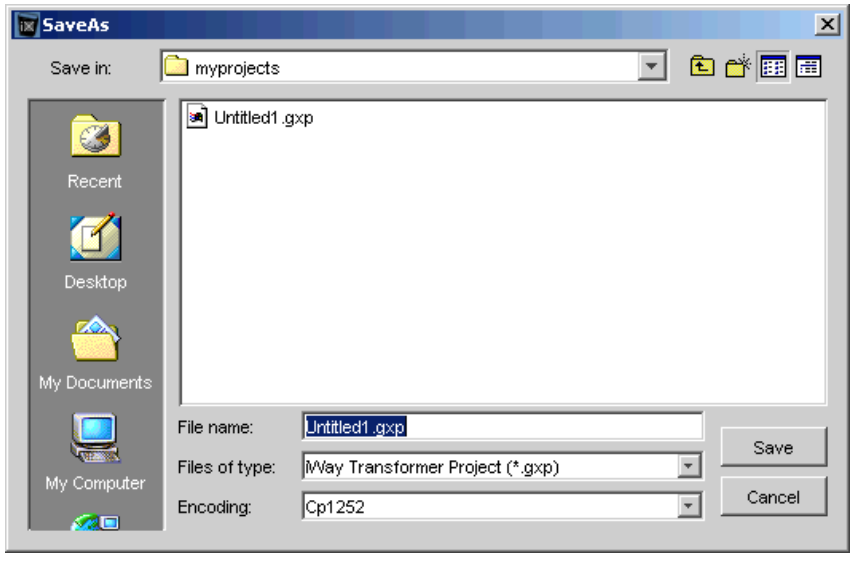
The following procedure explains how to save a newly created project.

Procedure: How to Save a Project

To save a newly created project:

1. From the File menu, select *Save As*.

   A dialog box opens as shown in the following image.



   a. Browse the file system to choose a location and file name for the project.

   b. If desired, define encoding.

      The default value is the system file encoding, which you can configure as follows: From the Tools menu, select *Options*, click the Encoding tab, select a value from the Default File encoding drop-down list, and then click *OK*.

2. Click *Save*.

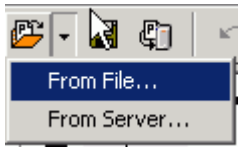
Note: To save a project any time during a work session, select *Save Project* from the File menu or click the *Save Project* button on the toolbar.

Opening Existing Projects

You can open an existing project from either a file system or a server. For information on configuring a server, see *Managing Server Connections* in Chapter 2, *Getting Started*.

Procedure: How to Open an Existing Project From a File System

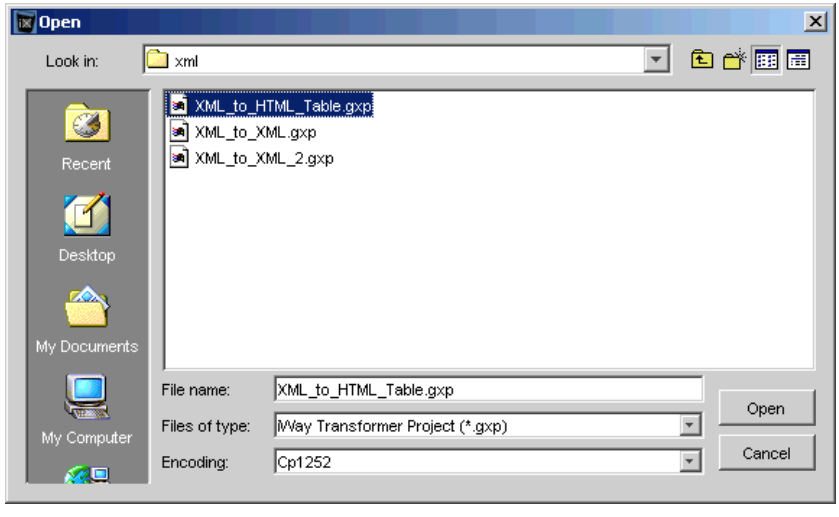
The following image shows the options after you select *Open Project*.



To open an existing project from a file system:

1. From the File menu, select *Open Project,* and then *From File.*

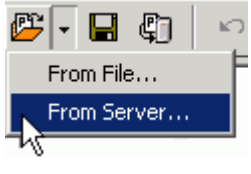
   The Open window opens as shown in the following image.



2. Browse through your file system and select the iWay Transformer project file you want to open.
3. Click *Open*.

Procedure: How to Open an Existing Project From a Server

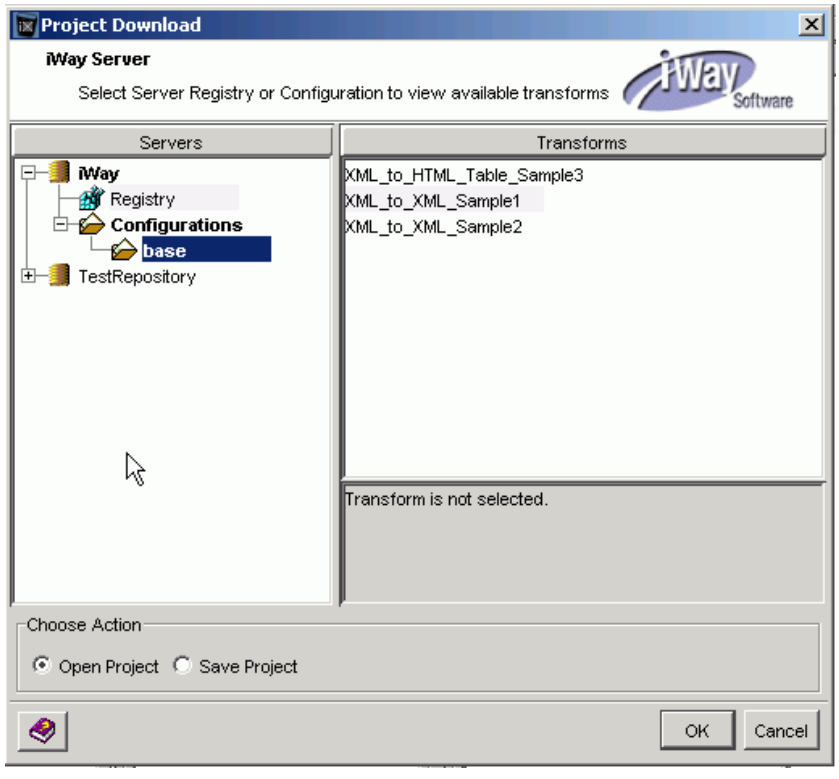
The following image shows the options after you select *Open Project*.



To open an existing project from a server:

1. From the File menu, select *Open Project*, and then select *From Server*.

   The Project Download dialog box opens as shown in the following image.



   a. Expand the Registry node or any Configuration node and select an iWay Transformer project file.

   b. In the Choose Action pane, select *Open Project*.

2. Click *OK*.

   Note: The Republish option is enabled and available for use.

Opening Sample Projects

iWay Transformer is packaged with many sample projects that you can use as reference when creating your own transformation projects.

To open a sample project, follow the instruction in *Opening Existing Projects* on page 3-11 and browse to the following directory:

`iWay55sm\tools\transformer\samples\sample_projects`

Sample projects reside in the following subdirectories:

- cdf
- context
- csv
- edi_ hipaa
- edi x12
- edifact
- fwf (fixed width)
- html output
- idoc
- loop
- swift
- xml

The directory name corresponds to the input or output data format of the sample projects contained within.

Testing Transformations

iWay Transformer provides the following options for testing transformations as you design them:

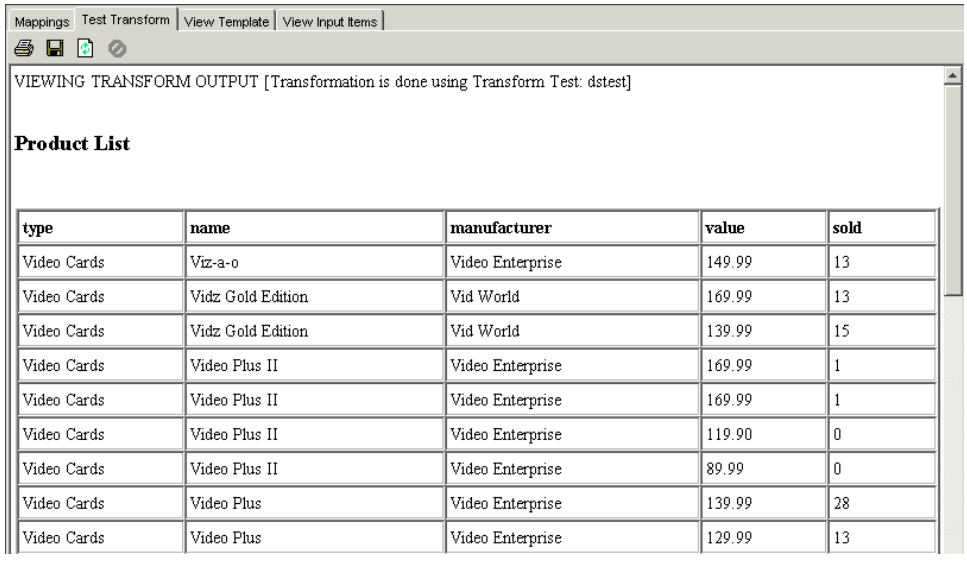
- **Single test**. The transformation is run against a single input file, and the result appears in the Test Transform tab. For information on how to run a single test, see *How to Test a Single Transformation* on page 3-14.
- **Batch test.** The transformation is run against a series of files and the result is stored in separate output files with indications of which transformations were successful. You can view these output files. For information on how to run a batch test, see *How to Test More Than One Transformation* on page 3-15.

Procedure: How to Test a Single Transformation

To test a single transformation:

1. From the Transform menu, select *Test*.

The transformation results appear in the Test Transform tab as shown in the following image.



type	name	manufacturer	value	sold
Video Cards	Viz-a-o	Video Enterprise	149.99	13
Video Cards	Vidz Gold Edition	Vid World	169.99	13
Video Cards	Vidz Gold Edition	Vid World	139.99	15
Video Cards	Video Plus II	Video Enterprise	169.99	1
Video Cards	Video Plus II	Video Enterprise	169.99	1
Video Cards	Video Plus II	Video Enterprise	119.90	0
Video Cards	Video Plus II	Video Enterprise	89.99	0
Video Cards	Video Plus	Video Enterprise	139.99	28
Video Cards	Video Plus	Video Enterprise	129.99	13

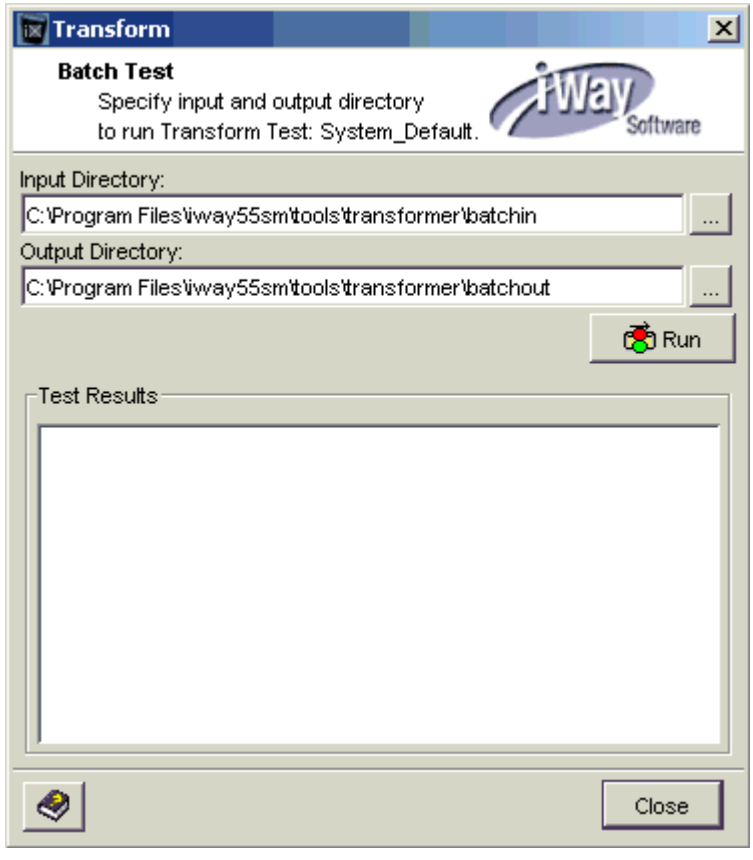
2. To save the test results to a file or print the test results, use the save and print icons found on the tab.

Procedure: How to Test More Than One Transformation

To perform a batch test:

1. From the Transform menu, select *Batch Test* or click the *Batch Test* icon found in the toolbar.

The Batch Test window opens as shown in the following image.



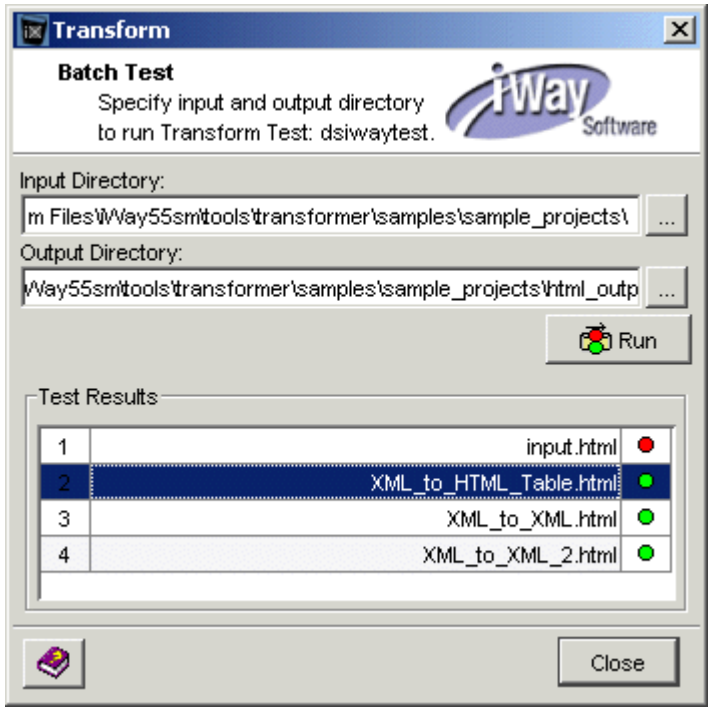
a. Type the path or browse to the directory for the input files.

b. Type the path or browse to the directory where you want the output files to be placed.

Note: The input and output directories must be different.

2. Click *Run*.

The path to the test results file appears in the Test Results pane as shown in the following image.

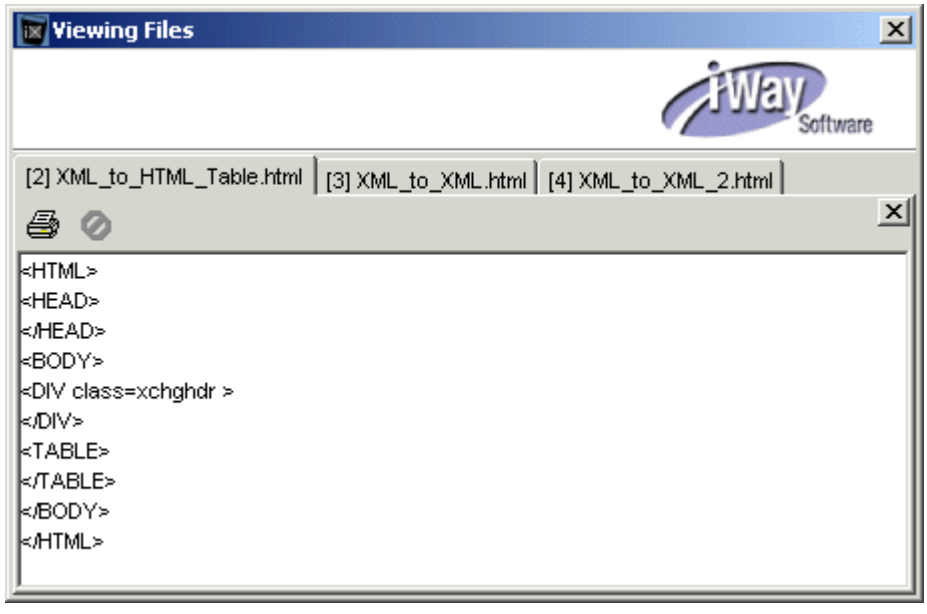


A green marker next to the file indicates the input file was retrieved successfully, or the output file was created successfully.

A red marker next to the file indicates that an error occurred during transformation.

3. Double-click the test results listing to see the content of the results file.

A Viewing Files window opens, as shown in the following image.



4. To print the file, click the printer icon on the tab.

Publishing Projects

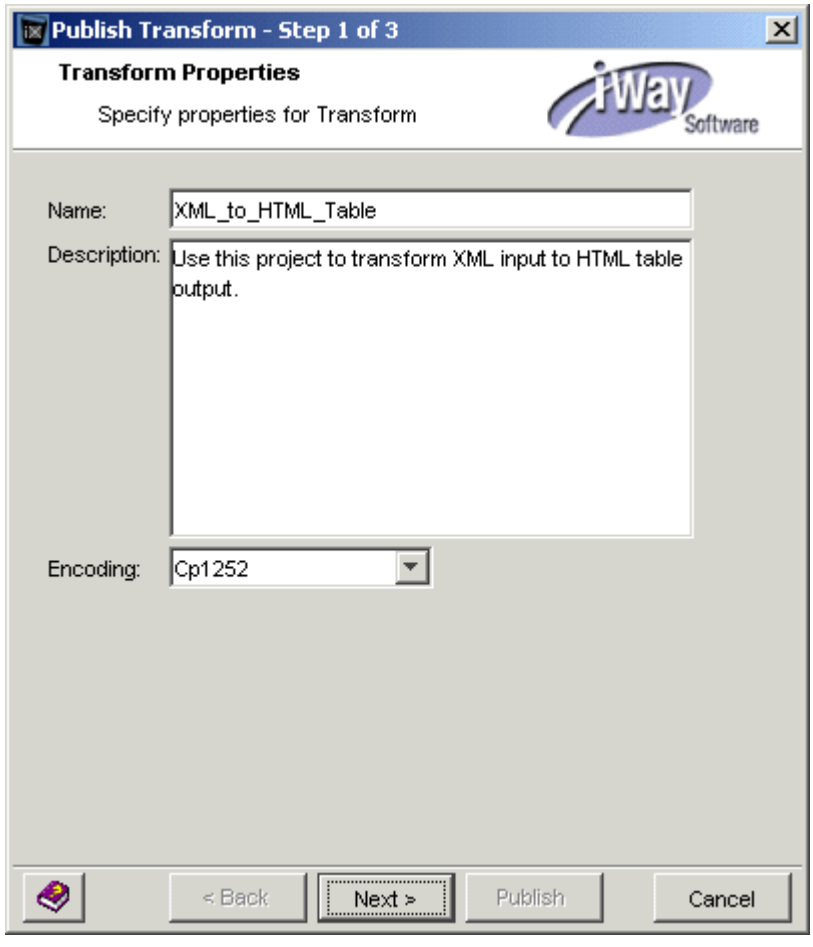
After you design and test your transformation, you can publish it so it is available by the iWay Service Manager run-time environment.

Procedure: How to Publish a Transformation Project

To publish a transformation project:

1. From the File menu, choose *Publish Project*.

The following image shows the Publish Transform Wizard that opens to the Transform Properties dialog box.



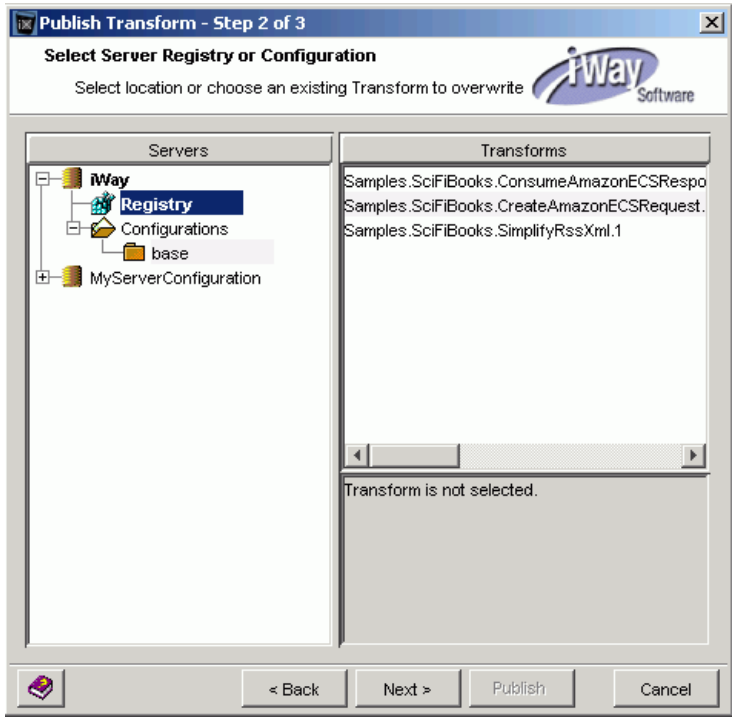
a. In the Name field, type a name for the project.

b. In the Description field, type a description.

c. From the Encoding drop-down list, choose a different value to change the encoding.

2. Click *Next*.

The Select Server Registry or Configuration dialog box opens as shown in the following image.



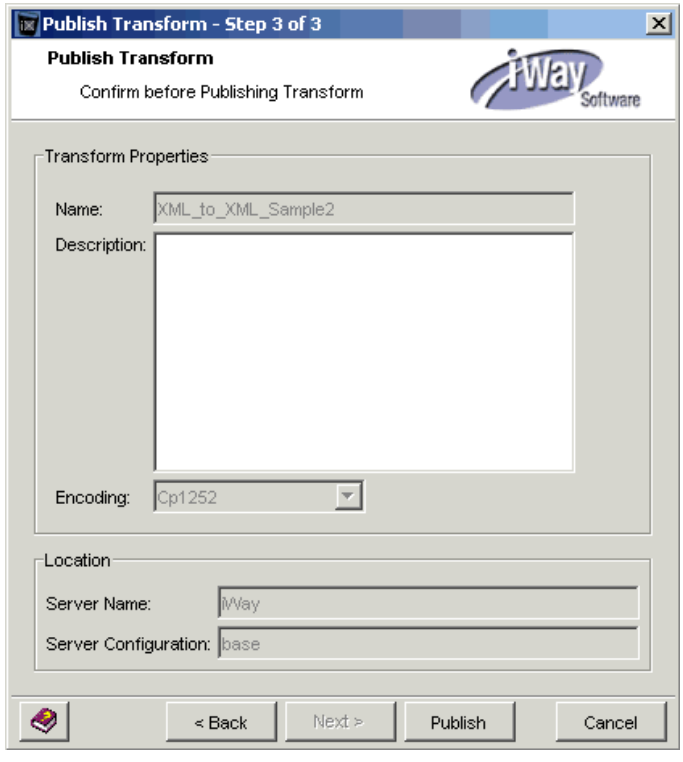
a. In the Servers pane, expand a server node, and then select the configuration to which you want to publish your project.

The Transforms pane lists the transforms in the selected folder.

b. To replace an existing project in the Transforms pane, select the transform you want to replace.

3. Click *Next*.

The Publish Transform dialog box opens, where you can confirm your project details before publishing. This dialog box contains two panes: Transform Properties and Location.



4. Click *Publish*.

A confirmation message appears once the operation is successful.

Using Templates

This topic describes iWay Transformer templates. iWay Software recommends using projects, instead of templates, to manage your transformations.

An iWay Transformer template is an XML configuration file that contains the rules of a transformation. The Transformation Engine processes input data and builds the output data according to these rules. Each transformation project requires its own template file. The template file defines the:

- Input and output data format.
- Structure of the output data.
- Operations required to build output data.

Note: A template file has a .xch extension.

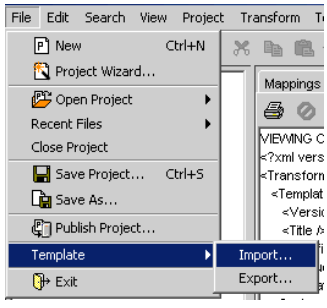
Viewing Templates

With your project open, you can select the Template option from the View menu. The template file that appears cannot be edited in the View Template tab. It is for viewing purposes only.

The following image shows a template on the View Template tab.

You can also import an existing template or export a template from a current transformation project. To import or export an existing template, select the Template option from the File menu and click the Import or Export option from the context menu.

The following image shows the File menu and the context menu that becomes available after you select the Template option.



After you choose either the Import or Export option, an Open or Save As dialog box opens to import or export your template.

CHAPTER 4

Configuring Input Items

Topics:

- Input Structures
- Configuring Input Data
- Embedding Content Structure in a Transformation Project

When creating an iWay Transformer project, you must specify the format of your input data and load the data structure into the project. This section describes how to specify the input data formats and structures.

Input Structures

An input structure defines exactly how the input data file is constructed. The input structure is loaded into the project and appears in the Input Items Structure pane of the Mappings tab. The input structure forms the basis for designing your output file.

An input structure can, but need not be, derived directly from the input data source:

- For XML input, the input structure comes from an *input structure file*. This is the DTD or XSD that describes the structure of your input data file.
- For CDF, IDoc, CSV, and Fixed Width input, the input structure comes from a special CDF, IDoc, CSV, or Fixed Width structure file.

  Sample files are provided with iWay Transformer. On Windows, the default location of these files is:

  ```
  C:\Program Files\iWay55\tools\transformer\samples\sample_projects
  ```

  For more information on how to process CDF files, see Appendix C, *Sample Transformations*.
- For SWIFT and EDI (X12, HIPAA, HL7, and EDIFACT) input, the input structure comes from a dictionary file.

  Dictionary files are special structure files that equate EDI or SWIFT to XML. For more information, see Appendix B, *Using Dictionary Files*.

The following table lists and describes the icons displayed in the Mappings tab to represent input items.

Icon	Description
	Element - used for all input data formats.
	Attribute - used for XML input only; provides an attribute to an XML parent tag.

Viewing the Input Structure

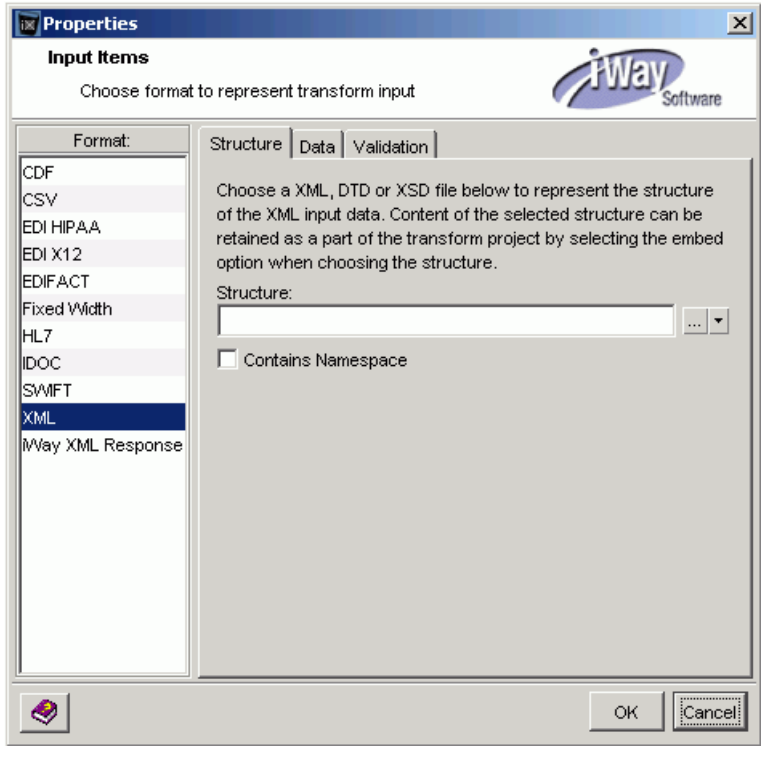
To view the input structure, you select the Input Files option from the View menu. This option is available only after you load an input structure. After the structure is loaded, you can view both the input structure and the input data files by selecting the appropriate button at the top of the pane.

Configuring Input Data

Input data is configured for use in iWay Transformer through the Input Items Properties dialog box. You can access this dialog box using one of the following methods.

- Double-clicking the Structure bar in the Input items pane.
- Right-clicking the Structure bar in the Input items pane and selecting the Input Items Properties option.
- Selecting the Input Items option from the Project menu.
- Right-clicking the Input Items option in the navigation pane and selecting Properties.

The following image shows the Input Items Properties dialog box which consists of a list of formats in the Formats pane on the left and three tabs on the right.



iWay Transformer activates the tabs depending on the input format selected in the Format list. The three tabs are:

- Structure
- Data
- Validation

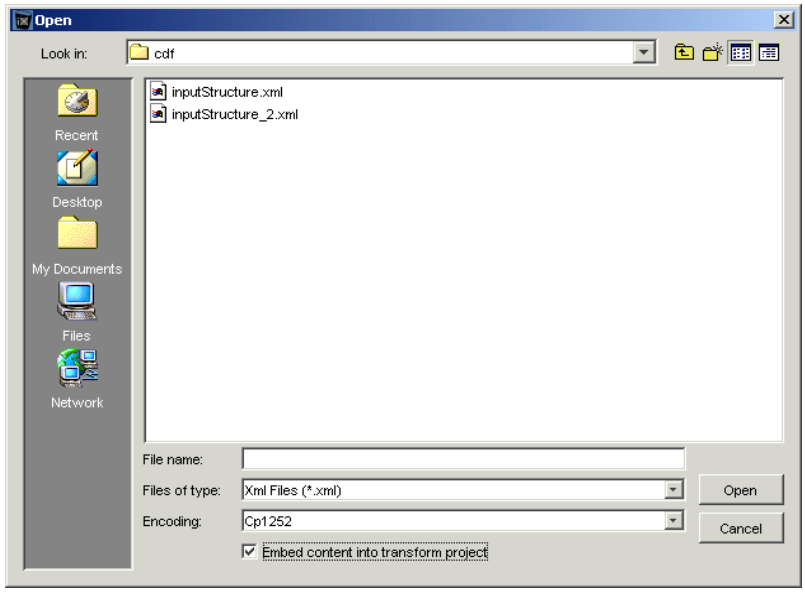
Embedding Content Structure in a Transformation Project

In the Structure tab, you specify an input structure file by either typing the name of the file or clicking the browse button to locate the file. In case of XML and iWay XML response transformations, there are additional options to load schema by importing it from the server registry, a server configuration, or from one of the available iWay Web services.

Note: No external Web services are supported, only iWay Web services, which are also known as iWay Business Services.

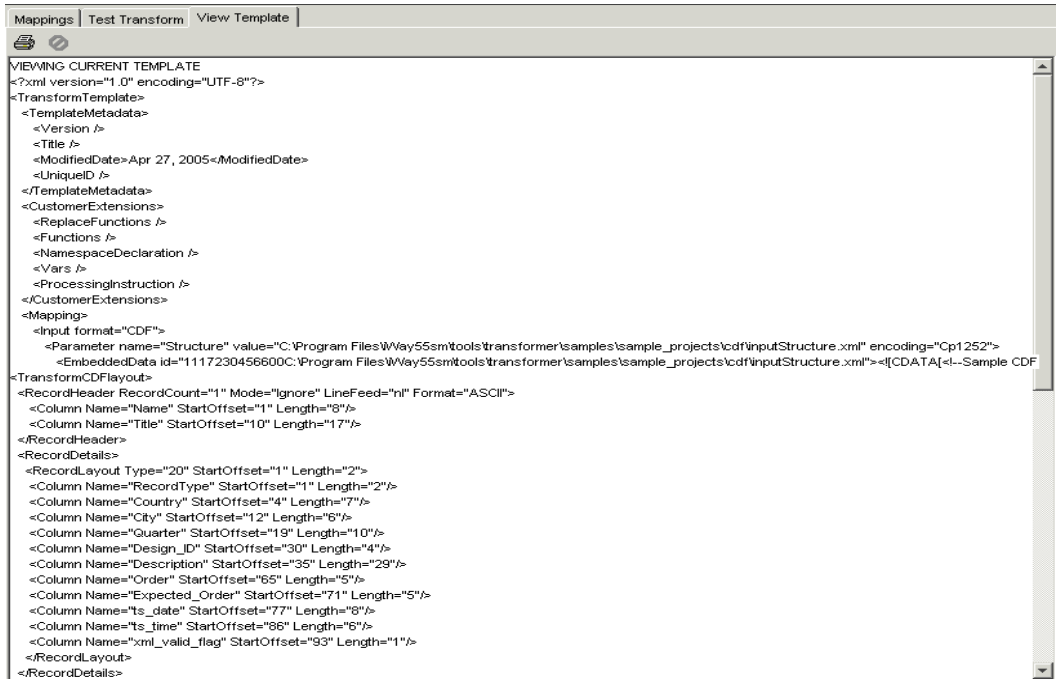
When you specify an input structure, you have the option to embed the content in your transformation project. Embedding the structure content ensures that your transformation always uses this embedded structure, even if the input structure on disk changes at some later point. This enables you to easily distribute the transformations you design across the enterprise.

The following image shows the window that opens when you click the browse button in the Structure tab. In addition to fields for the file name, there are drop-down lists for the type of file and encoding.



To embed the content, you select the check box next to Embed content into transform project. The check box is not selected by default.

The following image shows the View Template pane after the content is embedded. The content of the structure file is included in the Mappings section.



Reference: CDF Input Properties

The following table lists and describes the tabs and fields for the CDF input format. For more information on how to process CDF files, see Appendix C, *Sample Transformations*.

Field	Description
Structure tab	

Structure	Identifies the input structure source to be used in the project.
	To locate the file, type the file name or click the browse button.
	The following is a sample structure file, inputStructure.xml, which is included in the samples provided with the product:
	`<!--Sample CDF Structure -->`
	`<TransformCDFlayout>`
	`<RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl" Format="ASCII">`
	`<Column Name="Name" StartOffset="1" Length="8"/>`
	`<Column Name="Title" StartOffset="10" Length="17"/>`
	`</RecordHeader>`
	`<RecordDetails>`
	`<RecordLayout Type="20" StartOffset="1" Length="2">`
	`<Column Name="RecordType" StartOffset="1" Length="2"/><Column Name="Country" StartOffset="4" Length="7"/>`
	`<Column Name="City" StartOffset="12" Length="6"/>`
	`<Column Name="Quarter" StartOffset="19" Length="10"/>`
	`<Column Name="Design_ID" StartOffset="30" Length="4"/>`
	`<Column Name="Description" StartOffset="35" Length="29"/>`
	`<Column Name="Order" StartOffset="65" Length="5"/>`
	`<Column Name="Expected_Order" StartOffset="71" Length="5"/>`
	`<Column Name="ts_date" StartOffset="77" Length="8"/>`
	`<Column Name="ts_time" StartOffset="86" Length="6"/>`
	`<Column Name="xml_valid_flag" StartOffset="93" Length="1"/>`
	`</RecordLayout>`
	`</RecordDetails>`
	`</TransformCDFlayout>`
	To embed this structure into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.

Field	Description
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Decode data content	To decode data content from EBCDIC format, you select this check box. This check box is not selected by default.

Reference: CSV Input Properties

The following table lists and describes the tabs and fields for the CSV input format.

Field	Description
Structure tab	
Structure	Identifies the input structure source to be used in the project. This can be either the input data file itself or a file that has an identical structure to your intended input data file. To locate the file, type the file name or click the browse button. To embed this structure into your project, you click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Delimiter	You specify the delimiter character in this field. The default value is a comma (,). Although CSV stands for comma-separated values, iWay Transformer can process files that use other delimiter characters that otherwise correspond to the CSV format.

Field	Description

Header is included in the data	You select this check box if header is included in data. This check box is selected by default.
	
	The input.csv sample file provided with iWay Transformer includes a header with the data. The header is in the first line. The following is an excerpt from this sample file:
	
	`Country,"Province-Territory Name","Population (in 1000's)","Area Size (in km2)","Type"`
	
	`Canada,"Alberta","3097.5","661185","Province"`
	`Canada,"British Columbia","4092.8","948596","Province"`
	`.`
	`.`
	`.`

Reference: EDI Input Properties

The following table lists and describes the tabs and fields for the EDI input formats: EDI HIPAA, EDI X12, and EDIFACT.

Field	Description
Structure tab	
Dictionary Header	To locate the file, type the file name or click the browse button. If you leave this field blank, default header information based on the respective EDI X12, EDI HIPAA, or EDIFACT standard will be provided for you. To embed this content into your project, you click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.

Field	Description
Dictionary	A dictionary file is an XML representation of your input and is required to generate the input structure. For more information, see Appendix B, *Using Dictionary Files*. To locate the file, type the file name or click the browse button. To obtain the dictionaries, you must apply the HIPAA package from the iWay Service Manager console. For more information, see the *iWay Format Adapter for HIPAA User's Guide*. To embed this dictionary into your project, you click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Segment Delimiter	Specifies the character that indicates the end of a segment. If you leave this field blank, iWay Transformer uses a line return character.
Element Delimiter	Specifies the character that indicates the end of an element. If you leave this field blank, iWay Transformer uses a line return character.
Component Element Delimiter	Specifies the character that indicates the end of an a component element. If you leave this field blank, iWay Transformer uses a line return character.

Field	Description
Escape Character	In this field, you specify the escape character to use when reading the input data. If you leave this field blank, iWay Transformer uses: • For EDI HIPAA and EDI X12, the character specified in the ISA segment. • For EDIFACT, the character specified in the UNA segment. If no such segment exists in the transformation file, iWay Transformer uses: • For EDI HIPAA and EDI X12, a backslash (\). • For EDIFACT, a question mark (?).
Validation tab: To validate EDI input of the transformation run time, you choose from the following options.	
None	To skip validation, you select this option. This option button is selected by default.
Use validation rule defined by the project input structure	To validate the transformation using the rule defined by the project input structure, you select this option. This option is not selected by default. For EDI HIPAA and EDI X12, selecting this option activates the Ignore NTE Segment check box.
Ignore NTE Segment	For EDI HIPAA and EDI X12, select this check box to ignore the NTE segment. This check box is not selected by default. This check box is not available for EDIFACT.

Reference: Fixed Width Input Properties

The following table lists and describes the tabs and fields for the Fixed Width input format.

Field	Description
Structure tab	

Structure	Identifies the input structure source to be used in the project.
	
	To locate the file, type the file name or click the browse button. The following is a sample structure file:
	
	`<?xml version="1.0"?>`
	`<TransformCDFlayout>`
	`<RecordHeader>`
	`<Column Name="DESCRIPTION" StartOffset="1" Length="10" />`
	`</RecordHeader>`
	`<RecordDetails>`
	`<Loop ID="Loop1" Req="M" Min="1" Max="10">`
	`<RecordLayout Type="10" StartOffset="1" Length="2">`
	`<Column Name="L1_10-C1" StartOffset="3" Length="5" />`
	`<Column Name="L1_10-C2" StartOffset="8" Length="5" />`
	`</RecordLayout>`
	`<RecordLayout Type="20" StartOffset="1" Length="2">`
	`<Column Name="L1_20-C1" StartOffset="3" Length="10" />`
	`<Column Name="L1_20-C2" StartOffset="13" Length="10" />`
	`</RecordLayout>`
	`</Loop>`
	`<Loop ID="Loop2" Req="M" Min="1" Max="10">`
	`<RecordLayout Type="30" StartOffset="1" Length="2">`
	`<Column Name="L2_30-C1" StartOffset="3" Length="5" />`
	`<Column Name="L2_30-C2" StartOffset="8" Length="5" />`
	`</RecordLayout>`
	`<RecordLayout Type="40" StartOffset="1" Length="2">`
	`<Column Name="L2_40-C1" StartOffset="3" Length="10" />`
	`<Column Name="L2_40-C2" StartOffset="13" Length="10" />`
	`</RecordLayout>`
	`</Loop>`
	`</RecordDetails>`
	`</TransformCDFlayout>`
	
	To embed this structure into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.

Field	Description
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Record Delimeter	To specify delimeter options from an input data file.
Trim Columns	Selected by default, this check box is used to trim columns from the input data file.

Reference: HL7 Input Properties

The following table lists and describes the tabs and fields for the HL7 input format.

Field	Description
Structure tab	
Data Type Dictionary	To locate the file, type the file name or click the browse button. To embed this dictionary into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Dictionary Header	To locate the file, type the file name or click the browse button. If you leave this field blank, default header information based on the HL7 standard will be supplied for you. To embed this content into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Dictionary	A dictionary file is an XML representation of your input and is required to generate the input structure. To locate the file, type the file name or click the browse button. To embed this content into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.

Field	Description
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Segment Terminator	In this field, you specify the character that indicates the end of a segment.

Reference: IDOC Input Properties

The following table lists and describes the tabs and fields for the IDOC input format.

Field	Description
Structure tab	

Field	Description

Structure	Identifies the IDOC input structure source to be used in the project.
	
	To locate the file, type the file name or click the browse button.
	
	The following is an excerpt from a sample structure file, structure.txt, which is included in the samples provided with the product.
	
	`BEGIN_RECORD_SECTION`
	`BEGIN_CONTROL_RECORD`
	`BEGIN_FIELDS`
	`NAME TABNAM`
	`TEXT Name of table structure`
	`TYPE CHARACTER`
	`LENGTH 000010`
	`FIELD_POS 0001`
	`BYTE_FIRST 000001`
	`BYTE_LAST 000010`
	
	`NAME MANDT`
	`TEXT Client`
	`TYPE CHARACTER`
	`LENGTH 000003`
	`FIELD_POS 0002`
	`BYTE_FIRST 000011`
	`BYTE_LAST 000013`
	`.`
	`.`
	`.`
	
	To embed this content into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.

Data tab	
Data File	To locate the file, type the file name or click the browse button.

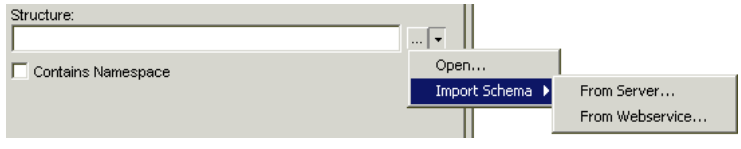
Reference: SWIFT Input Properties

The following table lists and describes the tabs and fields for the SWIFT input format.

Field	Description
Structure tab	
Dictionary	A dictionary file is an XML representation of your input and is required to generate the input structure. For more information, see Appendix B, *Using Dictionary Files*. To locate the file, you type the file name or click the browse button. To obtain the dictionaries, you must apply the Swift package from the iWay Service Manager console. For more information, see the *iWay Format Adapter for Swift User's Guide*. To embed this content into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Data tab	
Data File	To locate the file, type the file name or click the browse button.

Reference: XML and iWay XML Response Input Properties

The following table lists and describes the tabs and fields for the XML input formats: XML and iWay XML Response.

Field	Description
Structure tab	
Structure	Identifies the XML, schema, or DTD file that is the input structure source to be used in the project. This can be either the input data file itself or a file that has an identical structure to your intended input data file. To locate the file, you type the file name or click the browse button. The iWay 5.5 SM version of Transformer offers additional options to load schema allowing better integration with iWay Service Manager.  You can load schema from the server registry or from a server configuration by selecting *Import Schema*, *From Server*. Another option is to load schema from one of the available iWay Web services by selecting *Import Schema*, *From Webservice*. **Note:** No external Web services are supported, only iWay Web services, which are also known as iWay Business Services. In the Import Schema dialog box, type the Service Provider URL and select the service to display schemas available for selection in the left pane. To embed this content into your project, click the browse button to access the Open window and then select the check box next to Embed content into transform project. For more information, see *Embedding Content Structure in a Transformation Project* on page 4-4.
Contains Namespace	Select this check box if the input contains a namespace. This check box is not selected by default.

Field	Description
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Validation tab: To validate XML input of the transformation run time, you choose from the following options.	
None	To skip validation, you select this option. This option button is selected by default.
Use validation rule defined in the input data	To validate the transformation using the rule defined in the input data, select this option. This option is not selected by default.
Use user-defined DTD or XSD file	To validate the transformation using a DTD or XSD file, select this option. To locate the file, type the file name or click the browse button.

CHAPTER 5

Configuring Output Items

Topics:

- Output Structure
- Configuring Output Data
- Specifying Output Item Mapping
- Parent Items
- Parent Properties
- Output Item Filter
- Building and Altering Output Structures

When creating a transformation project, you must specify the output structure and assign values to the output items. The output structure can also be loaded into the project.

The following topics describe how to specify the output data structure.

Output Structure

The output structure specifies exactly how the output will be constructed. The Output structure uses a parent/child system similar to XML to produce *output blocks* in the output file (see *Parent Items* on page 5-22). An output block is one loop of output data in the output file for a parent and its children. Because parents can be nested within parents for some output data formats, output blocks can contain output blocks.

It is the output structure, not the actual output data, that you design and that appears in the Output Items Structure pane of the Mappings tab. You can design the output structure in the Output Items Structure pane using structure components or pre-load it from a specific file (or database information). After the structure is defined, you must assign values for the output data, as described in *Specifying Output Item Mapping* on page 5-20.

The Output Items Structure pane of the Mappings tab displays icons to represent output items. The following table lists and describes the icons and includes an image of each.

Icon	Name	Description
	Parent	Used for all output data formats.
	Child	Used for all output data formats.
	Element	Used for all output data formats.
	Attribute	Used for XML output only; provides an attribute to an XML parent tag.
	Comment	Used for all output data formats; supply a comment.
	Content	Used for all output data formats; provides content (value) for a parent item.
	CDATA	Used to indicate sections that you want the XML parser to ignore during validation. CDATA sections can include special characters that are not parsed.
	iWay XML Request or iWay Embedded XML Request	Used only for iWay XML request or iWay embedded XML request output; enables you to embed an SQL query or stored procedure.

Viewing the Output

To view output, you can run a test transformation on your project at any time by using the Test Transform feature. For instructions on running a test transformation, see Chapter 3, *Working with Projects*.

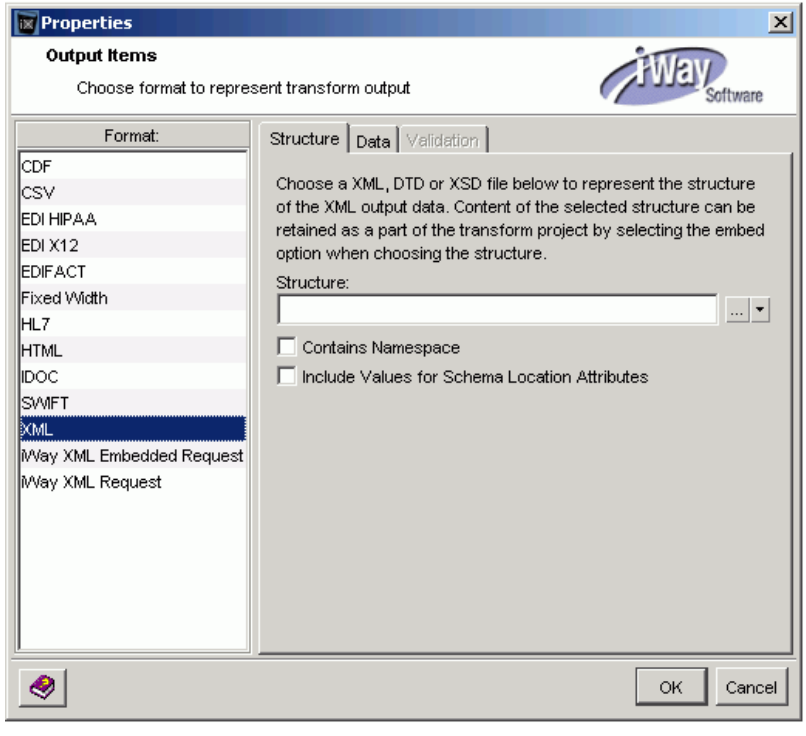
Configuring Output Data

The Output Items Properties dialog box provides the parameters and options to configure the output data for your project.

You can access this dialog box using one of the following methods.

- Double-clicking the Structure bar in the Output Items pane.
- Right-clicking the Structure bar in the Output Items pane and selecting Output Items Properties.
- Selecting Output Items from the Project menu.
- Right-clicking Output Items in the navigation pane and selecting Properties.

The following image shows the Output Items Properties dialog box which consists of a list of formats in the Formats pane on the left and three tabs on the right.



iWay Transformer activates the tabs depending on the output format selected in the Format list. The three tabs are:

- Structure
- Data
- Validation

Reference: CDF Output Properties

The following table lists and describes the tabs and fields for the CDF output format. For more information on how to process CDF files, see Appendix C, *Sample Transformations*.

Field	Description
Structure tab	

Structure	Identifies the output structure source to be used in the project.
	To locate the file, type the file name or click the browse button.
	The following is a sample structure file, inputStructure.xml, which is included in the samples provided with the product:
	`<!--Sample CDF Structure -->`
	`<TransformCDFlayout>`
	`<RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl" Format="ASCII">`
	`<Column Name="Name" StartOffset="1" Length="8"/>`
	`<Column Name="Title" StartOffset="10" Length="17"/>`
	`</RecordHeader>`
	`<RecordDetails>`
	`<RecordLayout Type="20" StartOffset="1" Length="2">`
	`<Column Name="RecordType" StartOffset="1" Length="2"/><Column Name="Country" StartOffset="4" Length="7"/>`
	`<Column Name="City" StartOffset="12" Length="6"/>`
	`<Column Name="Quarter" StartOffset="19" Length="10"/>`
	`<Column Name="Design_ID" StartOffset="30" Length="4"/>`
	`<Column Name="Description" StartOffset="35" Length="29"/>`
	`<Column Name="Order" StartOffset="65" Length="5"/>`
	`<Column Name="Expected_Order" StartOffset="71" Length="5"/>`
	`<Column Name="ts_date" StartOffset="77" Length="8"/>`
	`<Column Name="ts_time" StartOffset="86" Length="6"/>`
	`<Column Name="xml_valid_flag" StartOffset="93" Length="1"/>`
	`</RecordLayout>`
	`</RecordDetails>`
	`</TransformCDFlayout>`

Data tab	
Padding	Specify the character to use for padding empty spaces. If you leave this parameter blank, iWay Transformer uses a blank space for padding.
Align	Specifies the alignment of the output. Choose either Left or Right alignment. The default value is Left.

Field	Description
Encode	To encode your data in EBCDIC format, select this check box. By default, this check box is not selected.

Reference: CSV Output Properties

The following table lists and describes the tabs and fields for the CSV output format.

Field	Description
Structure tab	
Structure	Identifies the output structure source to be used in the project. To locate the file, type the file name or click the browse button.
Data tab	
Delimiter	Specify the delimiter character. The default value is a comma (,). Although CSV stands for comma-separated values, iWay Transformer can process files that use other delimiter characters that correspond to the CSV format.
Header is included in the data	To include header in the data, select this check box. This check box is selected by default.
Always Add Quotes	To add quotation marks, select this check box. This check box is not selected by default.

Reference: EDI Output Properties

The following table lists and describes the tabs and fields for the EDI output formats: EDI HIPAA, EDI X12, and EDIFACT.

Field	Description
Structure tab	
Dictionary Header	To locate the file, type the file name or click the browse button. If you leave this field blank, default header information based on the respective EDI X12, EDI HIPAA, or EDIFACT standard is supplied for you.

Field	Description
Dictionary	A dictionary file is an XML representation of your output and is required to generate the output structure. For more information, see Appendix B, *Using Dictionary Files*. To locate the file, type the file name or click the browse button.
Data tab	
Data File	To locate the file, type the file name or click the browse button.
Segment Delimiter	Specifies the character that indicates the end of a segment. If you leave this field blank, iWay Transformer uses a line return character.
Element Delimiter	Specifies the character that indicates the end of an element. If you leave this field blank, iWay Transformer uses a line return character.
Component Element Delimiter	Specifies the character that indicates the end of an a component element. If you leave this field blank, iWay Transformer uses a line return character.
Escape Character	Specify the escape character to use for output data. If you leave this field blank, iWay Transformer uses: • For EDI HIPAA and EDI X12, the character specified in the ISA segment. • For EDIFACT, the character specified in the UNA segment. If one of the preceding named segments does not exist in the transformation file, iWay Transformer uses: • For EDI HIPAA and EDI X12, a backslash (\). • For EDIFACT, a question mark (?).
Validation tab: Choose from the following options to validate EDI output of the transformation run time.	
None	To skip validation, select this option. This option is selected by default.

Field	Description
Use validation rule defined by the project's output structure	To validate the transformation using the rule defined by the project output structure, select this option. This option is not selected by default. For EDI HIPAA and EDI X12, selecting this option activates the Ignore NTE Segment check box.
Ignore NTE Segment	For EDI HIPAA and EDI X12, select this check box to ignore the NTE segment. This check box is not selected by default. This check box is not available for EDIFACT.

Reference: Fixed Width Output Properties

The following table lists and describes the tabs and fields for the Fixed Width output format.

Field	Description
Structure tab	

Structure	Identifies the output structure source to be used in the project.
	To locate the file, type the file name or click the browse button.
	The following is a sample structure file:
	`<?xml version="1.0"?>`
	`<TransformCDFlayout>`
	`<RecordHeader>`
	`<Column Name="DESCRIPTION" StartOffset="1" Length="10" />`
	`</RecordHeader>`
	`<RecordDetails>`
	`<Loop ID="Loop1" Req="M" Min="1" Max="10">`
	`<RecordLayout Type="10" StartOffset="1" Length="2">`
	`<Column Name="L1_10-C1" StartOffset="3" Length="5" />`
	`<Column Name="L1_10-C2" StartOffset="8" Length="5" />`
	`</RecordLayout>`
	`<RecordLayout Type="20" StartOffset="1" Length="2">`
	`<Column Name="L1_20-C1" StartOffset="3" Length="10" />`
	`<Column Name="L1_20-C2" StartOffset="13" Length="10" />`
	`</RecordLayout>`
	`</Loop>`
	`<Loop ID="Loop2" Req="M" Min="1" Max="10">`
	`<RecordLayout Type="30" StartOffset="1" Length="2">`
	`<Column Name="L2_30-C1" StartOffset="3" Length="5" />`
	`<Column Name="L2_30-C2" StartOffset="8" Length="5" />`
	`</RecordLayout>`
	`<RecordLayout Type="40" StartOffset="1" Length="2">`
	`<Column Name="L2_40-C1" StartOffset="3" Length="10" />`
	`<Column Name="L2_40-C2" StartOffset="13" Length="10" />`
	`</RecordLayout>`
	`</Loop>`
	`</RecordDetails>`
	`</TransformCDFlayout>`

Data tab	
Record Delimeter	Specifies delimeter options in the output data file.

Field	Description
Column Padding	Specify the character to use for padding empty spaces. If you leave this parameter blank, iWay Transformer uses a blank space for padding.
Text Alignment	Specifies the alignment of the output. Choose either Left or Right alignment. The default value is Left.
Trim Text	Selected by default, this check box is used to trim text in the output data file.

Reference: HL7 Output Properties

The following table lists and describes the tabs and fields for the HL7 output format.

Field	Description
Structure tab	
Data Type Dictionary	To locate the file, type the file name or click the browse button.
Dictionary Header	To locate the file, type the file name or click the browse button. If you leave this field blank, default header information based on the HL7 standard is provided.
Dictionary	A dictionary file is an XML representation of your output and is required to generate the output structure. To locate the file, type the file name or click the browse button.
Data tab	
Escape Character	Specify the escape character to use for output data. The default value is a backslash (\).
Segment Terminator	Specify the character that indicates the end of a segment.
Field Separator	Specify the character to use to separate fields. The default value is a pipe symbol (|).
Component Separator	Specify the character to use to separate components. The default value is a caret(^).

Field	Description
Repetition Separator	Specify the character to indicate repetition. The default value is a tilde (~).
Subcomponent Separator	Specify the character to use to separate subcomponents. The default value is an ampersand (&).
Retain Empty Parent Tags	To retain empty parent tags, select this check box. This check box is selected by default.

Reference: HTML Output Properties

The following table lists and describes the tabs and fields for the HTML output format.

Field	Description
Data tab	
Mode	Specifies the format of the HTML output. Choose either Form or Table format.
Stylesheet Type	Specifies the format of the style sheet you are using. Choose text/xsl or text/css.
Stylesheet File	Identifies the stylesheet to apply to your output. To locate the file, type the file name or click the browse button. This file must match the selected file type.

Header	Type the header information you want to include in the output file. You can specify one or more lines, separated by a pipe symbol (|), for example,
	`This information|is inserted|into the output`
	adds the following header to your output:
	`<!--This information-->`
	`<!--is inserted-->`
	`<!--into the output-->`

Field	Description

Footer	Type the footer information you want to include in the output file. You can specify one or more lines, separated by a pipe symbol (|), for example,
	`This information|is inserted|into the output`
	adds the following footer to your output:
	`<!--This information-->`
	`<!--is inserted-->`
	`<!--into the output-->`

Reference: IDOC Output Properties

The following table lists and describes the tabs and fields for the IDOC output format.

Field	Description
Structure tab	

Field	Description

Structure	An IDOC dictionary file specifies exactly how the fields in the SAP IDOC file are arranged and is required to generate the output structure.
	
	To locate the file, type the file name or click the browse button.
	
	The following is an excerpt from a sample structure file, structure.txt, which is included in the samples provided with the product:
	
	`BEGIN_RECORD_SECTION`
	`BEGIN_CONTROL_RECORD`
	`BEGIN_FIELDS`
	`NAME TABNAM`
	`TEXT Name of table structure`
	`TYPE CHARACTER`
	`LENGTH 000010`
	`FIELD_POS 0001`
	`BYTE_FIRST 000001`
	`BYTE_LAST 000010`
	
	`NAME MANDT`
	`TEXT Client`
	`TYPE CHARACTER`
	`LENGTH 000003`
	`FIELD_POS 0002`
	`BYTE_FIRST 000011`
	`BYTE_LAST 000013`
	`.`
	`.`
	`.`

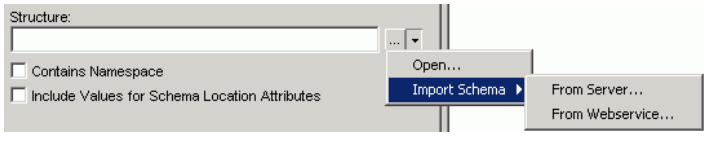
Reference: SWIFT Output Properties

The following table lists and describes the tabs and fields for the SWIFT output format.

Field	Description
Structure tab	
Dictionary	A dictionary file is an XML representation of your output and is required to generate the output structure. For more information, see Appendix B, *Using Dictionary Files*. To locate the file, type the file name or click the browse button.

Reference: XML Output Properties

The following table lists and describes the tabs and fields for the XML output formats: XML and iWay XML Embedded Request.

Field	Description
Structure tab	
Structure	Identifies the XML, schema, or DTD file that represents the output structure source to be used in the project. Type the file name or click the *browse* button to locate the file. The iWay 5.5 SM version of Transformer offers additional options to load schema allowing better integration with iWay Service Manager.  You can load schema from the server registry or from a server configuration by selecting *Import Schema*, *From Server*. Another option is to load schema from one of the available iWay Web services by selecting *Import Schema*, *From Webservice*. **Note:** No external Web services are supported, only iWay Web services, which are also known as iWay Business Services. In the Import Schema dialog box, type the Service Provider URL and select the service to display schemas available for selection in the left pane.
Contains Namespace	Select this option to include a namespace attribute as a property of the output items.
Include Values for Schema Location Attributes	Select this option to map the *schemaLocation* attribute to its original value.
Data tab	

Field	Description

Node Indent	Used to format the output, it indicates the number of spaces to indent the node. The default value is 4.
	
	For example, if node indent is 0, then all elements are aligned at the margin as follows:
	
	`<a>`
	`<b>`
	`</b>`
	`<c>`
	`</c>`
	`</a>`
	
	If node indent is 4, all elements except root are indented 4 spaces as follows:
	
	`<a>`
	`    <b>`
	`    </b>`
	`    <c>`
	`    </c>`
	`</a>`

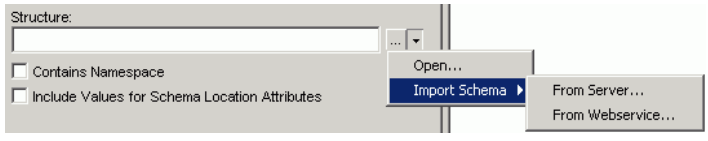
Optimization	Enables you to retain or omit empty elements and attributes. To retain empty elements and attributes, select *Do Not Optimize*. This is the default value. To omit empty elements, select *Remove Empty Parent* Tags. To omit empty elements and attributes, select *Remove All Empty Tags*.
Character Encoding	Choose the character encoding from the drop-down list. By default, UTF-8 is selected. To specify character encoding that is used for the input data, select *Same as input data* from the Character Encoding drop-down list.
Stylesheet Type	Choose either text/xsl or text/css.
Stylesheet File	To locate the style sheet file you want to use for the output, type the full path or click browse (...) to navigate to the file.
Header	Define the header information you want in the output.
Footer	Define the footer information you want in the output.

Field	Description
Validation tab: Choose from the following options to validate the XML output of the transformation run time.	
None	Option button, selected.
Use validation rule defined in the output data	Option button.
Use user-defined DTD or XSD file	Option button. To locate the file, type the file name or click the browse button.

Reference: iWay XML Embedded Request Output Properties

The following table lists and describes the tabs and fields for the iWay XML Embedded Request output properties.

Field	Description
Structure tab	

Field	Description
Structure	Identifies the XML, schema, or DTD file that represents the output structure source to be used in the project. Type the file name or click the *browse* button to locate the file. The iWay 5.5 SM version of Transformer offers additional options to load schema allowing better integration with iWay Service Manager.  You can load schema from the server registry or from a server configuration by selecting *Import Schema*, *From Server*. Another option is to load schema from one of the available iWay Web services by selecting *Import Schema*, *From Webservice*. **Note:** No external Web services are supported, only iWay Web services, which are also known as iWay Business Services. In the Import Schema dialog box, type the Service Provider URL and select the service to display schemas available for selection in the left pane.
Contains Namespace	Select this option to include a namespace attribute as a property of the output items.
Include Values for Schema Location Attributes	Select this option to map the *schemaLocation* attribute to its original value.
Data tab	

Field	Description

Node Indent	Used to format the output, it indicates the number of spaces to indent the node. The default value is 4.
	
	For example, if node indent is 0, then all elements are aligned at the margin as follows:
	
	`<a>`
	`<b>`
	`</b>`
	`<c>`
	`</c>`
	`</a>`
	
	If node indent is 4, all elements except root are indented 4 spaces as follows:
	
	`<a>`
	`    <b>`
	`    </b>`
	`    <c>`
	`    </c>`
	`</a>`

Optimization	Enables you to retain or omit empty elements and attributes. To retain empty elements and attributes, select *Do Not Optimize*. This is the default value. To omit empty elements, select *Remove Empty Parent* Tags. To omit empty elements and attributes, select *Remove All Empty Tags*.
Character Encoding	Choose the character encoding from the drop-down list. By default, UTF-8 is selected. To specify character encoding that is used for the input data, select *Same as input data* from the Character Encoding drop-down list.
Stylesheet Type	Choose either text/xsl or text/css.
Stylesheet File	To locate the style sheet file you want to use for the output, type the full path or click browse (...) to navigate to the file.
Header	Define the header information you want in the output.
Footer	Define the footer information you want in the output.

Reference: iWay XML Request Output Properties

The following table lists and describes the tabs and fields for the iWay XML Request output properties.

Field	Description
Data tab	

Node Indent	Used to format the output, it indicates the number of spaces to indent the node. The default value is 4.
	
	For example, if node indent is 0, then all elements are aligned at the margin as follows:
	
	`<a>`
	`<b>`
	`</b>`
	`<c>`
	`</c>`
	`</a>`
	
	If node indent is 4, all elements except root are indented 4 spaces as follows:
	
	`<a>`
	`    <b>`
	`    </b>`
	`    <c>`
	`    </c>`
	`</a>`

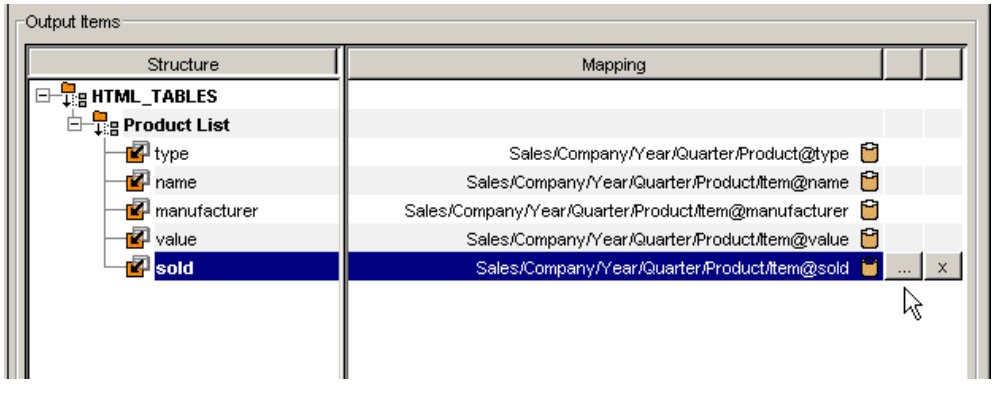
Optimization	Enables you to retain or omit empty elements and attributes. To retain empty elements and attributes, select *Do Not Optimize*. This is the default value. To omit empty elements, select *Remove Empty Parent* Tags. To omit empty elements and attributes, select *Remove All Empty Tags*.
Character Encoding	Choose the character encoding from the drop-down list. By default, UTF-8 is selected. To specify character encoding that is used for the input data, select *Same as input data* from the Character Encoding drop-down list.
Stylesheet Type	Choose either text/xsl or text/css.

Field	Description
Stylesheet File	To locate the style sheet file you want to use for the output, type the full path or click browse (...) to navigate to the file.
Header	Define the header information you want in the output.
Footer	Define the footer information you want in the output.

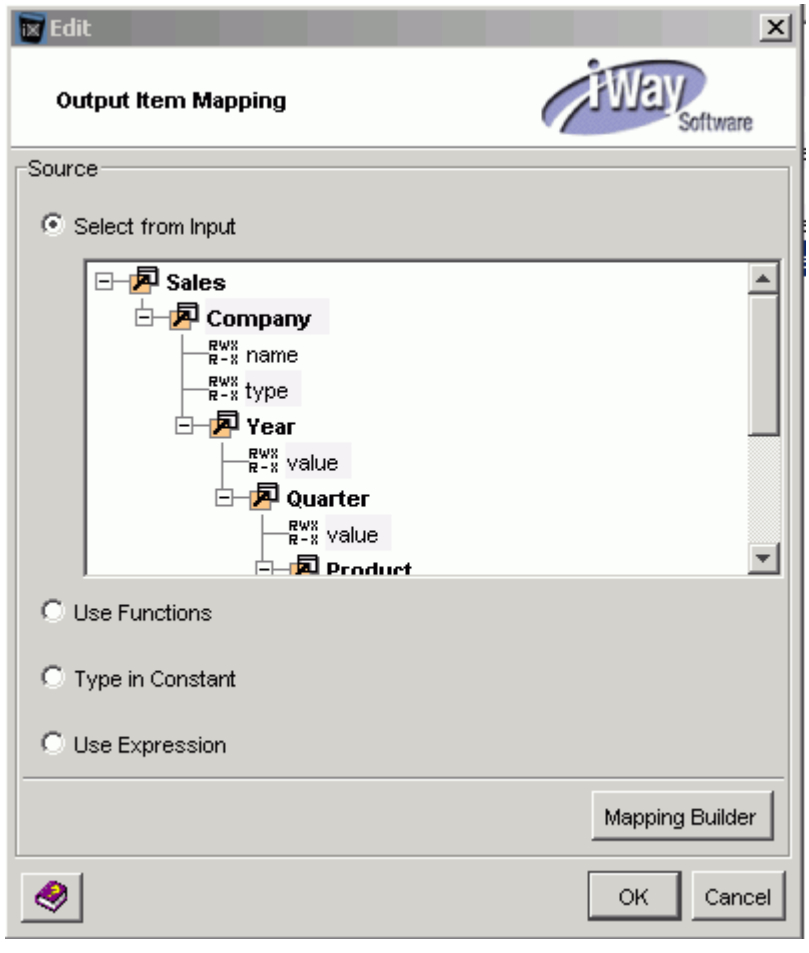
Specifying Output Item Mapping

To perform a transformation, creating only an output structure is not sufficient. You must specify the mapping for each output item.

You can specify the output item value in the Output Item Mapping dialog box. To access this dialog box, in the Mappings tab of the Output Items pane, you select an output item and then, you click the ellipsis button as shown in the following image.



The following image shows the Output Item Mapping dialog box that opens.



The following table lists and describes options you can select to set the output item value.

Option	Description
Select from Input	To point to an input item on which to base the value of the output item, for example: patients/personal_info/age.
Use Functions	To access a list of functions that operate on the value of input items, for example: @SUBTRACT('2001', 'patients/personal_info/yob'
Type in Constant	To type a value in the Constant field that opens, for example: 'Patient age is unavailable'

Option	Description

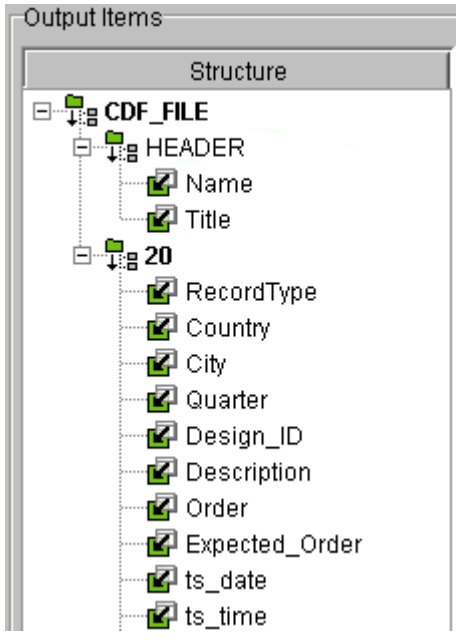
Use Expression	To set an expression in the Expression field that opens. Click the *browse* button to open the Expression Builder dialog box.
	
	An expression is a concatenation of any number of input items and constant values in any order, in the following format:
	
	`<input item>+'constant'+...`
	
	For example:
	
	`'For '+tree/fruit+tree/fruit/year+': '+tree/fruit/apple/yield+`
	`' bushels, '+tree/fruit/apple/avgweight+' grams avg.'`
	
	Note: It is strongly encouraged to use the @CONCAT function instead of an expression to concatenate strings. For more information on using the @CONCAT function and its properties, see Appendix A, *Pre-Defined Functions*.

In addition, you can click the Mapping Builder button to open the Mapping Builder. For more information, see Chapter 6, *About Functions*.

Parent Items

Every output structure, regardless of the output data format, has at least one output parent item. A parent item marks the start of an output block by specifying the start of an output loop. The first item in your output structure is always a parent item, created automatically by iWay Transformer when you create your initial output structure. The first parent item tells iWay Transformer to produce output by looping through the entire output structure, while reading the input file and performing the transformation.

Consider the following possible CDF output structure as shown in the following image.



The first output item, CDF_FILE, is the first parent item. It ensures that iWay Transformer loops over the entire output structure. The next item is HEADER, which is also a parent item. HEADER creates an output block which includes the child element items Name and Title. The item, 20, is also a parent item, creating an output block to include the next 10 child element items (RecordType to ts_time). Parent items are shown with the Parent icon. Element items are shown with the Element icon.

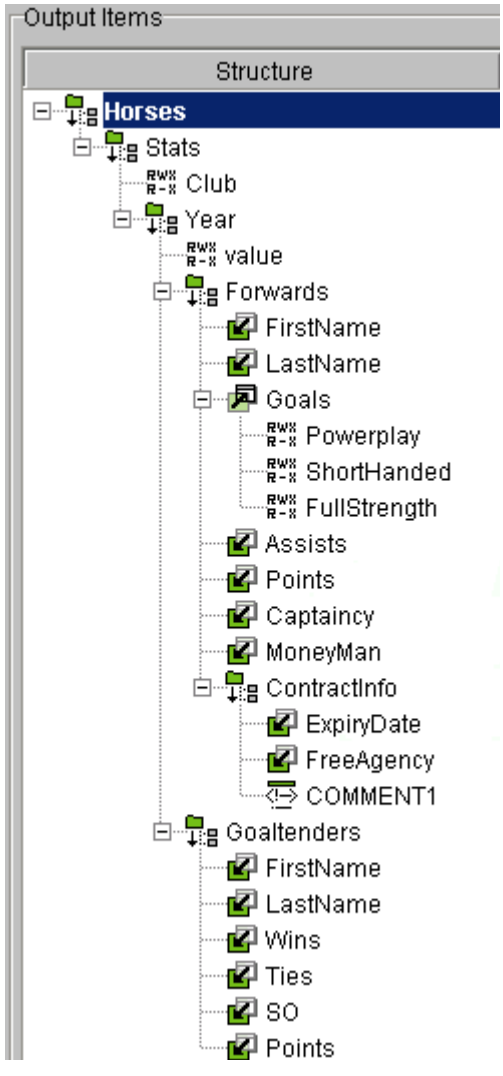
Additional XML Output Structures

In addition to indicating the start of an output loop, a parent item can also have the following structures depending on the output data format:

- XML output - first parent item value specifies document root. Other parent item values specify an XML parent tag name; can be used to create nested loops.
- EDI (X12, HIPAA, EDIFACT) output - parent item value specifies an XML parent tag name; can be used to create nested loops.

You can use parent items to create complex output structures in terms of parent tags and nested output loops for the XML output formats (XML and EDI). Remember that EDI (X12, HIPAA, and EDIFACT) output is treated as XML output due to the use of the EDI dictionary files.

The following image shows an example of an output structure that uses many nested output loops.



iWay Transformer makes it easy to create a complex output structure for your XML output file. The way you arrange the output structure in the Output Items Structure pane of the Mapping window is exactly how your output is structured.

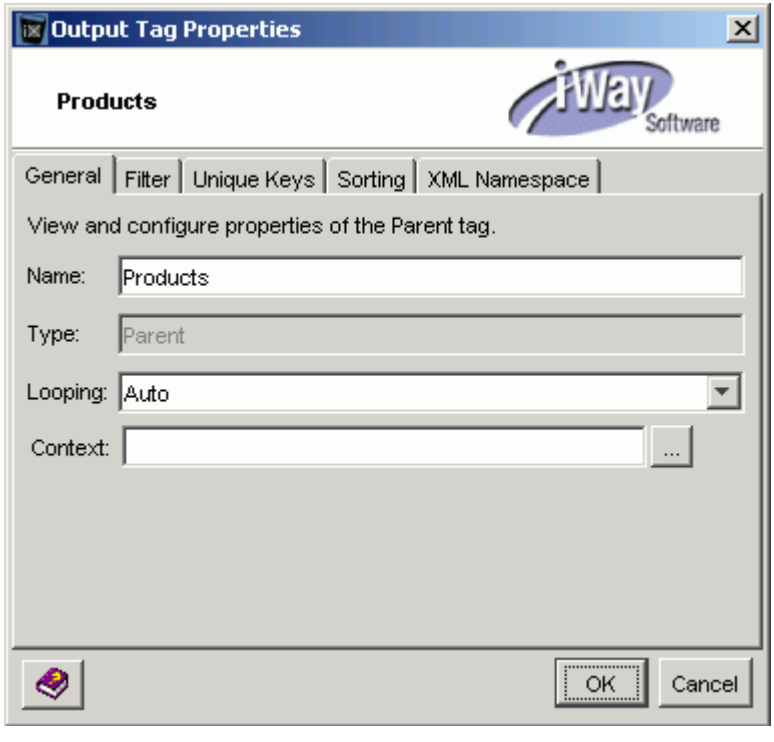
Parent Properties

You can set parent properties for a parent output tag. To view parent properties in the Structure tab of the Output Items pane, you must right-click a parent output item and then, select *Properties*. The Output Tag Properties dialog box opens.

The following topics describe the tabs available from the Output Tag Properties dialog box.

General Tab

The following image shows the Output Tag Properties dialog box. Below the tag name panel are five tabs: General, Filter, Unique Keys, Sorting, and XML Namespace. The dialog box defaults to the General tab. Notice that the Type field contains a default value of Parent, which indicates the type of the tag being used.



In order to decide how your parent element will loop, ensure that you have selected the correct value from the Looping drop-down list:

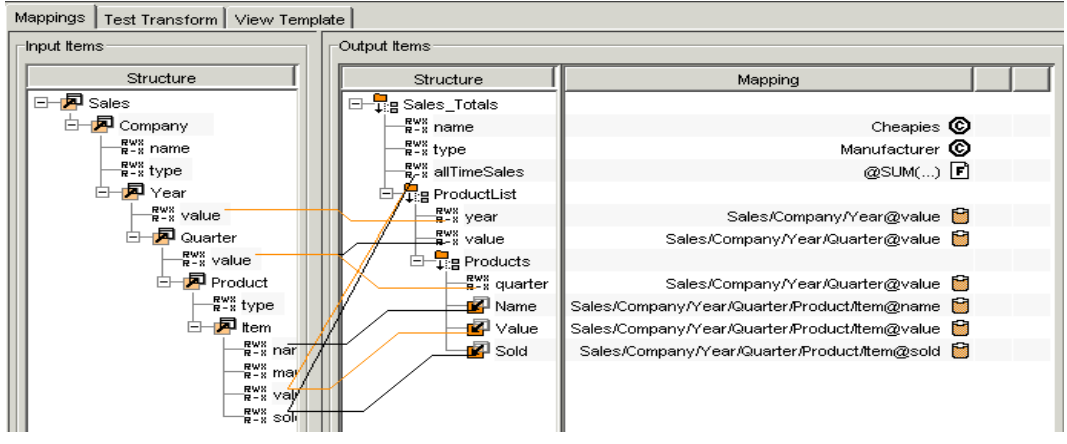
- **Auto** - Enforces the default looping mechanism.
- **True** - Indicates that element should loop.
- **False** - Indicates that element should not loop.

- **Agg** - Indicates that looping should be aggregate.

  If aggregate looping is selected for the parent element, all instances of that output parent item that have the same attribute values are combined into one (unique) parent item. Any items that are children of these elements are also amassed and made subordinate to the new parent element.

The Context field in the General tab is an advanced feature designed to help control local looping. Applying it to an item in your output structure can solve looping complexity issues. It is especially useful in projects involving changes in the hierarchy of a structure. The Context specifies the item location where the block of input that is being used for looping begins. Generally, this should be set to the innermost parent block, which encapsulates all of the data that must be used within this tag.

To better understand how the Context feature works, consider the sample project XML_to_XML_2 located in the /sample/sample_projects/xml/XML_to_XML_2 directory. The input to output structure mapping is shown in the following image.

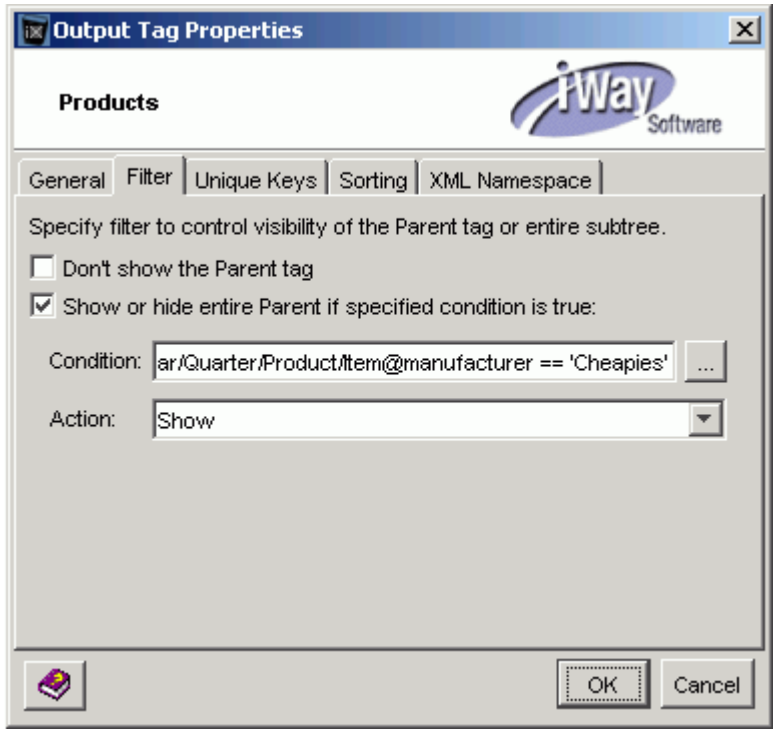


The input is an XML file that lists details for the sales of a company. The information is organized by year, quarter, product type, and individual item (brand). The output is an XML file containing the sales only for items with the manufacturer (Cheapies). The output sales are arranged by year, quarter, and item.

Note that Product List has looping set to aggregate, and context set to Sales/Company/Year. Aggregation is required to enforce that only one parent element will exist in the output for each individual quarter. The context is set to Year. Year is the lowest element in the input structure hierarchy that contains all of the information used within Product List and is the beginning of the block in which looping will occur.

Filter Tab

The following image shows the Filter tab of the Output Tag Properties dialog box.



The *Show or hide entire Parent if specified condition is true* checkbox enables you to do just that, by defining the condition and resulting action on the parent and therefore, on each individual output block produced by that output loop.

If this checkbox is selected and the Hide option is selected from the Action drop-down list, then that particular output block is not shown in the output file.

If the condition you enter is true for an output loop and the Show option is selected from the Action drop-down list, then only that output block is shown in the output file, and all others that do not match the condition are suppressed from the output file.

The condition you type must take the form of <inputfield> argument <'constant'>, for example:

```
Horses/Team/Years/Player/Goals == '30'
```

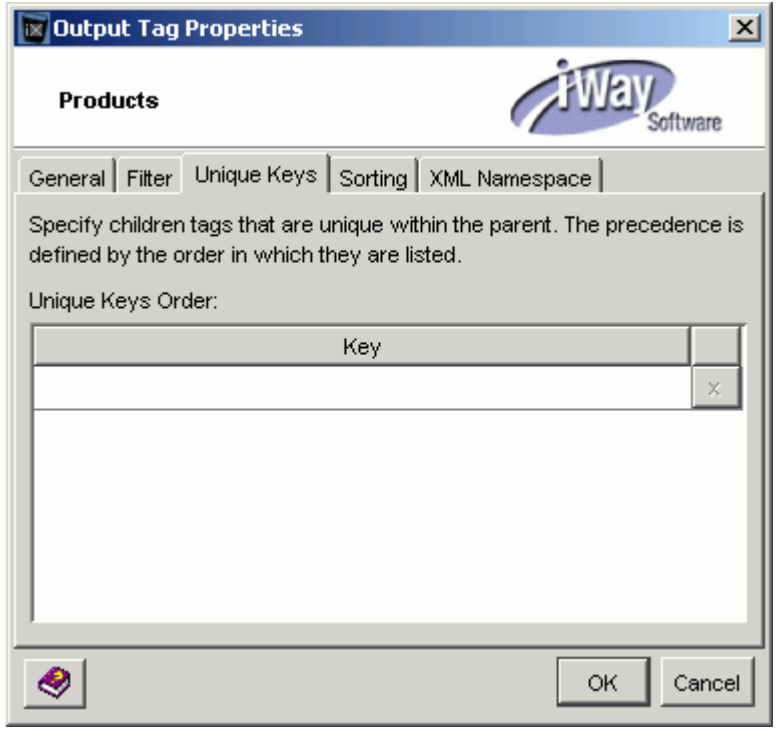
The following table lists the possible arguments for the block condition.

= =	equal to

!=	not equal to
>=	greater than or equal to
<=	less than or equal to
>	greater than
<	less than

Unique Keys Tab

The following image shows the Unique Keys tab of the Output Tag Properties dialog box.



You can control the uniqueness of a parent's output child items by defining items as unique in the Unique Keys tab. For example, assume that you generate the following non-unique XML output data:

```
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle</Name>
</Product>
<Product>
    <ProductID>SR71</ProductID>
    <Name>SR-71 Blackbird</Name>
</Product>
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle Again</Name>
</Product>
```

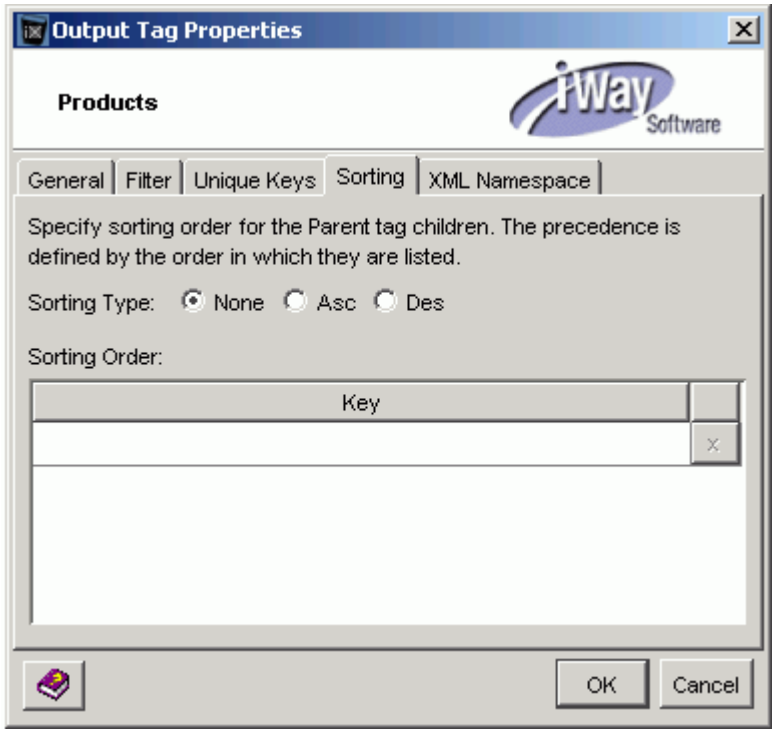
If you choose to make the output data unique by specifying the output item, ProductID, in the Unique Keys parent property, then your final output is the following:

```
<Product>
    <ProductID>SR71</ProductID>
    <Name>SR-71 Blackbird</Name>
</Product>
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle</Name>
</Product>
```

The second instance of the XR281 product was removed because it was not unique to the output item ProductID.

Sorting Tab

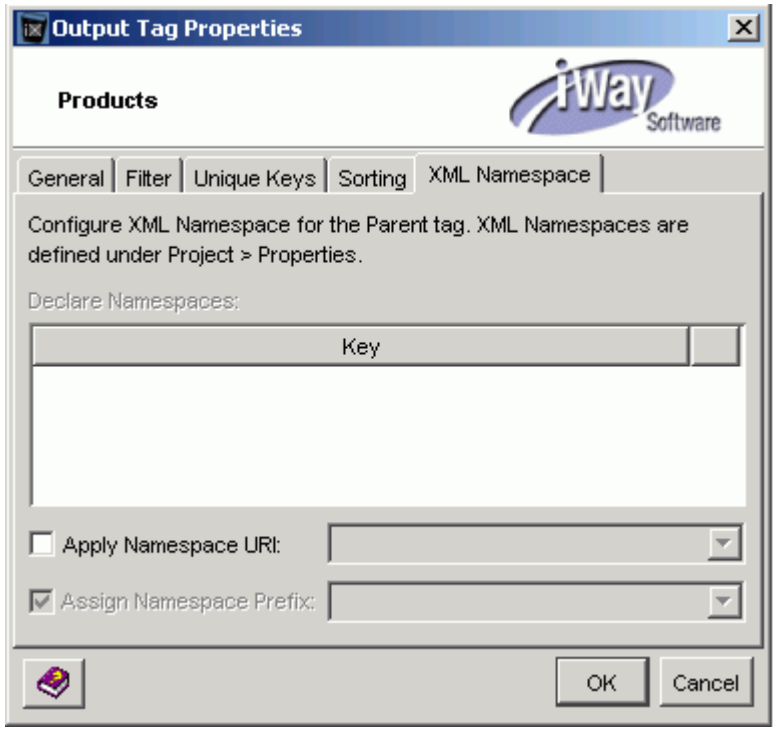
The following image shows the Sorting tab of the Output Item Tag Properties dialog box.



The Sorting tab enables you to choose a child element by which to sort the output. If you sort, the output is sorted in ascending or descending alphanumeric order by the element you choose. The Sort Type option specifies how the output is sorted. Possible values are None, Ascending, or Descending. Sort Type defaults to None.

XML Namespace Tab

The following image shows the XML Namespace tab of the Output Tag Properties dialog box.

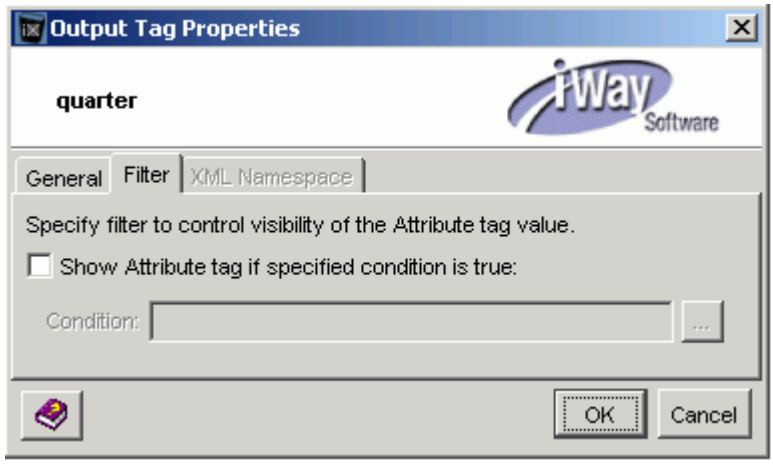


For more information on defining XML Namespaces, see *Configuring XML Namespaces* in Chapter 2, *Getting Started*.

Output Item Filter

You can set a filter for any non-parent output item or attribute, similar to the Filter parent tab. To set a filter for a non-parent output item, in the Output Items Structure pane of the Mapping window, right-click the output item and select *Properties*.

Once the Output Tag Properties dialog box opens, navigate to Filter tab as shown in the following example:



The Filter tab enables you to set a condition on an output item when the *Show Attribute tag if specified condition is true* checkbox is selected. If the condition you enter in the condition field is true, as one loop of the input data is processed, the output item of that particular loop through the input data is populated in the output file.

The condition you type must take the form of <inputfield> = = <'constant'>, for example:

```
Horses/Team/Years/Player/Goals == '30'
```

Building and Altering Output Structures

For most output formats, you can load an existing output structure file into your project. The output structure file's structure then appears in the Output Items Structure pane. The exception is HTML, for which you must build the output structure for your project within the Output Items Structure pane using iWay Transformer tools. Also, most output structures can be altered once in the iWay Transformer Output Items Structure pane. But there are those that should never be altered. The rules for altering output structures of specific data formats are:

Do *not* alter the output structure in the Output Items Structure pane if your output is:

- CDF
- EDI X12
- EDI HIPAA
- EDIFACT
- IDOC

All of these output data formats require a mandatory structure file (dictionary file for EDI outputs). The output structure must match the dictionary file or structure file exactly. If you alter the output structure in the Output Items Structure pane of the Mapping screen for any of these outputs, even by one character, your transformation will not work properly.

You can alter the output structure if your output format is:

- CSV
- HTML
- XML

As noted previously, you must build your output structure using structure components in iWay Transformer for the HTML output format.

Adding New Output Items

One method of adding new output items is structure mapping. For more information about structure mapping, see *Parent Items* on page 5-22. Another method of adding new output items is by right-clicking an existing output item and selecting Add followed by the type of item you wish to add.

The following table lists and describes the available output items and includes an image of the output item icon.

Icon	Name	Description
	Parent	Used for all output data formats.
	Child	Used for all output data formats.
	Element	Used for all output data formats.
	Attribute	Used for XML output only; provides an attribute to an XML parent tag.
	Comment	Used for all output data formats; supply a comment.
	Content	Used for all output data formats; provides content (value) for a parent item.
	CDATA	Used to indicate sections that you want the XML parser to ignore during validation. CDATA sections can include special characters that are not parsed.

Icon	Name	Description
	iWay XML Request or iWay Embedded XML Request	Used only for iWay XML request or iWay embedded XML request output; enables you to embed an SQL query or stored procedure.

The type of output item you can add depends on the output item that you right-clicked and the output data format. For example, if you right-click an XML element output item, you can only add an attribute. Through such restrictions, the integrity of the output structure is maintained.

CHAPTER 6

About Functions

Topics:

- Writing Custom Functions
- Defining Custom Functions
- Defining Replace Functions
- Using JDBC Replace Functions
- Using the Mapping Builder
- Using the @JDBCLOOKUP Function

A function is a piece of Java code written to perform a calculation upon, or manipulation of, input data. You can apply functions to produce the output value within its output item mapping dialog.

iWay Transformer provides pre-defined functions but also enables you to define your own. All custom functions created for use with iWay Transformer must be written in Java and stored in the custom_functions directory.

This section explains how to define and write custom functions and how to use the replace function.

Writing Custom Functions

The following instructions assume that you are familiar with the Java programming language.

Custom functions must be written in Java and must be stored in the custom_functions directory to be available for use with iWay Transformer. The custom function you design is a subclass of the super class AbstractFunction.

You must implement a constructor for your class and the two abstract methods execute() and getReturnType() that are declared in AbstractFunction. You also may choose to add your own methods. The Java code for your custom function has a format similar to the following:

```
import com.iwaysoftware.transform.common.function.AbstractFunction;
public class MY_FUNCTION_NAME extends AbstractFunction
{
     public MY_FUNCTION_NAME()
     {
        setName("MY_FUNCTION_NAME");
        setDescription("This is what my function does...");
     }
     public Object execute() throws Exception
     {
        // perform function's execution here
        // and return desired output value
     }
     public Class getReturnType()
     {
        return MY_FUNCTION'S_RETURN_TYPE;
     }
}
```

Import Statement

You must have the following statement as the top line of your custom function implementation:

```
import com.iwaysoftware.transform.common.function.AbstractFunction;
```

This enables you access to methods that you require in your function implementation.

Class Declaration

As stated previously, a custom function is a subclass of the class AbstractFunction. Therefore, you declare your custom function as follows:

```
public class MY_FUNCTION_NAME extends AbstractFunction {
...
}
```

Constructor

The only methods you must call within your constructor are setName(String name) and setDescription(String description), which are pre-defined in the AbstractFunction class. The values you type as the parameters in these methods are used only when you access your custom function through iWay Transformer.

The function name used in iWay Transformer is the parameter that you set in the setName method, and the function description used by iWay Transformer is the message you type as the parameter in setDescription.

execute( )

The method execute() is one of two abstract methods declared in the AbstractFunction class. Your custom function, a subclass of AbstractFunction, must therefore, implement this method. execute() is declared as follows:

```
public Object execute() throws Exception {
...
}
```

The body of this method is written as you would write any method in Java.

Getting Arguments

Retrieving arguments passed to your custom function is performed through the methods getValueInstance(), getArgument(int index), and getFunctionContext(), which are pre-defined in the functions package that your custom function imported. To store two arguments in String variables argOne and argTwo, write the following:

```
String argOne =
(String)getFunctionContext().getArgument(0).getValueInstance();

String argTwo =
(String)getFunctionContext().getArgument(1).getValueInstance();
```

Other methods that you may find useful in your implementation are getNumberOfArguments() and getArguments(). The first returns an integer indicating the number of arguments passed to your function by the user and is used in conjunction with method, getFunctionContext(), as follows:

```
int num = getFunctionContext().getNumberOfArguments();
```

getArguments() returns an array containing all of the arguments passed to your function by the user. To store your arguments in an array of Objects called myArray, write the following:

```
Object[] myArray = getFunctionContext().getArguments();
```

getReturnType( )

getReturnType() is the other abstract method declared in the AbstractFunction class. Your custom function, a subclass of AbstractFunction, must therefore, implement this method. The value returned by getReturnType() indicates the type of value you return in method execute(). The choices for your return value are:

- java.lang.Boolean.class
- java.lang.Character.class
- java.lang.Double.class
- java.lang.Float.class
- java.lang.Integer.class
- java.lang.Long.class
- java.lang.String.class
- java.lang.StringBuffer.class

Defining Custom Functions

When an iWay Transformer pre-defined function does not exist to perform the task you require, you can write your own custom function. Custom functions must be written in Java and compiled to create a .class file, which must then be stored in the custom_functions directory to be available for use with iWay Transformer.

Before using a custom function within iWay Transformer, test it to ensure it works properly. For details on creating custom functions, see *Writing Custom Functions* on page 6-2.

Defining a custom function in iWay Transformer is equivalent to creating a pointer to the actual custom function stored in the custom_functions directory. Custom functions are defined through the Project Properties dialog box.

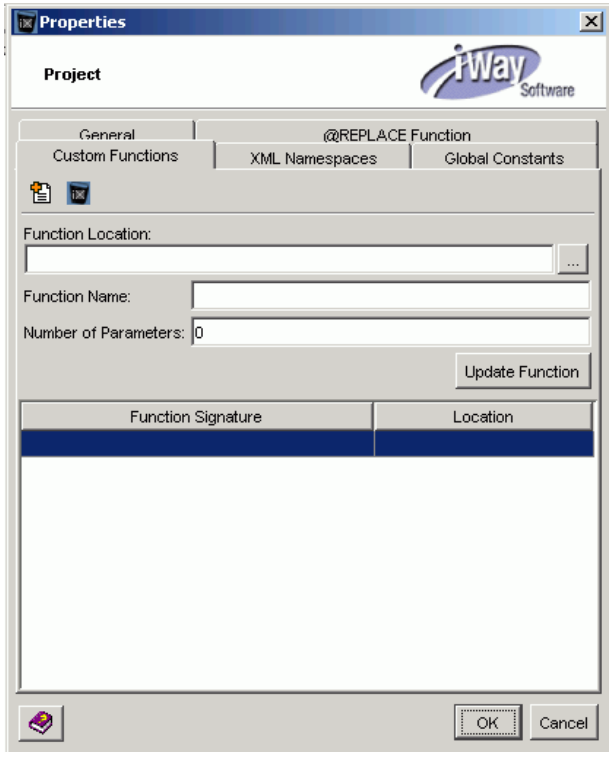
Procedure: How to Define a Custom Function

1. From the menu, select *Project* and then, *Properties*.

   This opens the Project Properties dialog box with the General tab showing.

2. To define a custom function, click the *Custom Functions* tab.

The following image shows the Project Properties dialog box with the Custom Functions tab showing.



a. In the Function Location, field provide the full or relative path of the location of the function, for example:

```
C:\Program Files\iWay55sm\tools\transformer\custom
functions\myFunction.jar
```

b. In the Function Name field, provide the fully-qualified Java name of the custom function, for example,

```
com.myCompany.mySoftware.MyClass.functionName.
```

c. In the Number of Parameters field, provide the number of parameters that the function accepts.

3. Click *Update Function*.

The custom function appears in the list with the function name in the Function Signature column, and the function location in the Location column. You can use your custom function to assign an output item value.

Compiling Your .java File

To make your custom function available for use with iWay Transformer, compile your completed Java code to create a .class file. To do so, make sure your .java file is located in the custom functions directory. Then, type the following at the command prompt from within this same directory:

```
%JAVA_HOME%\javac -classpath ..\iway55sm\lib\iwtranse.jar
FUNCTION_NAME.java
```

where:

`%JAVA_HOME%`

Is the directory in which your javac.exe is located, and FUNCTION_NAME is the name of your custom function.

Note: Before you compile your custom functions, verify that you have the iwtranse.jar file in your class path.

Sample Custom Function

The following Java code is the implementation for a function that returns a string resulting from adding 25 to the function input value. Comments have been added for readability.

```
import com.iwaysoftware.transform.common.function.AbstractFunction;
public class ADD25 extends AbstractFunction {
     public ADD25 {
        /* specifies the function name to be used within iAM */
        setName("ADD25");
        /* specifies the function description to be used within iAM */
       setDescription("Return a string resulting from adding 25 to input
value.");
     }
     public Object execute() throws Exception {
        /* retrieves the first argument and assigns it to Double variable
'input' */
        Double input =
(Double)getFunctionContext().getArgument(0).getValueInstance();
        /* creates a variable to hold the value that will get returned */
        double answer;
        answer = 25 + input.doubleValue();
        /* casts the value that will get returned to a String */
        String answerString = Double.toString(answer);
        return answerString;
     }
     public Class getReturnType() {
        /* specifies that method execute() returns a String */
        return java.lang.String.class;
     }
}
```

Migrating Custom Functions

Custom functions must be written in Java and must be stored in the custom_functions directory to be available for use with iWay Transformer. The custom function you design is a subclass of the super class AbstractFunction. You are required to implement a constructor for your class and the two abstract methods execute() and getReturnType() that are declared in AbstractFunction. You also may choose to add your own methods. In previous releases of iAT, the Java code for your custom function had a format similar to the following:

```
import com.xmlglobal.goxmltransform.engine.functions.*;
public class MY_FUNCTION_NAME extends AbstractFunction
{
public MY_FUNCTION_NAME()
{
setName("MY_FUNCTION_NAME");
setDescription("This is what my function does...");
}
public Object execute() throws Exception
{
// perform function's execution here
// and return desired output value
}
public Class getReturnType()
{
return MY_FUNCTION'S_RETURN_TYPE;
}
}
```

The following procedure describes how to migrate custom functions created from a previous iWay release, for example, 5.2.104, that are called by your modified .xch file.

Note: Before you compile your migrated custom functions, verify that you have the iwtranse.jar file in your class path.

Procedure How to Migrate Custom Functions

To migrate custom functions, modify your import statements as follows:

1. Comment out xmlglobal package(s), for example:

   ```
   //import com.xmlglobal.goxmltransform.engine.functions.*;
   ```

2. Add the iwaysoftware package, which must be imported, for example:

   ```
   import com.iwaysoftware.transform.common.function.AbstractFunction;
   ```

Using Custom Functions at Run Time

Custom functions must be published for use at run time. They must be placed in a compressed registered library or in the following iWay Service Manager class path:

```
iWay55sm\etc\manager\transformations\custom_functions
```

For more information on how to register a library, see the *iWay Service Manager User's Guide*.

Defining Replace Functions

Replace functions match and replace input data values. For example, suppose that your input data has a field called "car_age" with a value of either "new" or "used". You decide that you do not like the term "used" and want to use "pre-owned" instead. You can define a replace function with a match-replace pairing of match="used" and replace="pre-owned."

Replace functions work like custom functions in that you must first define the function and then, apply it in the output item mapping value definition that you want to affect. In the case of replace functions, do this through the @REPLACE function.

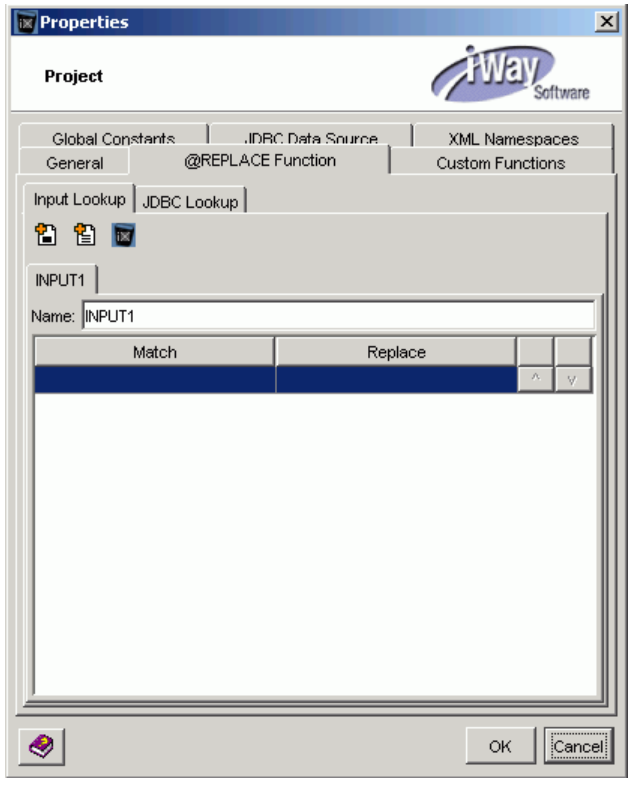
The process of defining a replace function accounts for setting up the values but not the mapping. You must map replace functions using the Mapping Builder.

Procedure: How to To Define a Replace Functions

To define a replace function:

1. From the menu, select *Project*, and then *Properties*.

The Project Properties dialog box opens with the General tab showing as shown in the following image.



2. To define replace functions, select the *@REPLACE Function* tab.

There are two types of replace functions: Input Lookup and JDBC Lookup.

3. To define replace functions in which you define the static match-replace values, use the Input Lookup tab.

By default, there is one initial replace function named INPUT1 on the Input Lookup tab. You can change the name to whatever you wish.

To define replace functions in which the match-replace values are taken from columns in a specified database table, enabling you to access input from different databases, use the JDBC Lookup tab.

This topic only describes Input Lookup replace functions. For information on JDBC replace functions, see *Using JDBC Replace Functions* on page 6-12.

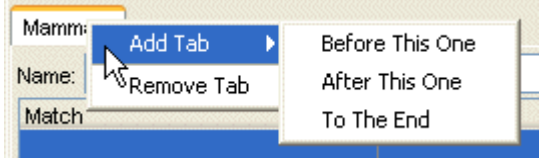
Procedure: How to Add a Replace Function

An Input Lookup replace function is represented by one tab within the Input Lookup tab.

To add and define a new replace function (tab):

1. Right-click the tab of an existing replace function.
2. Select *Add Tab*.

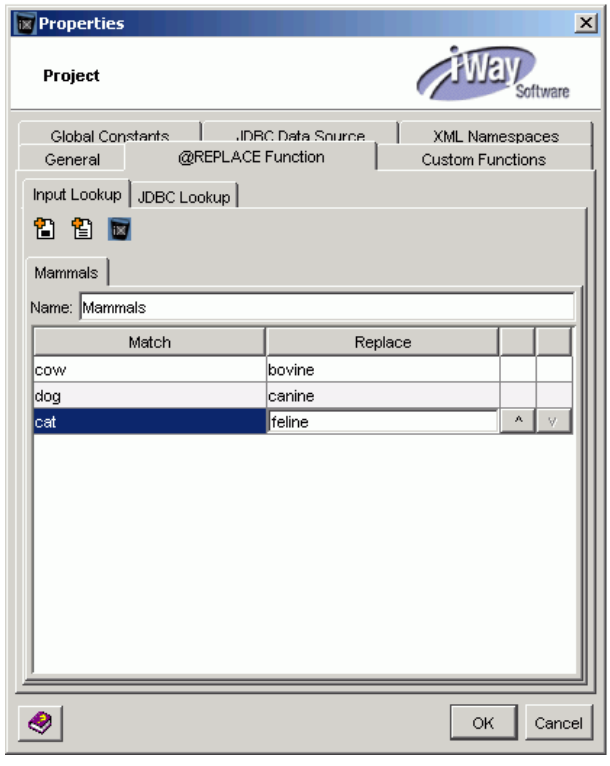
   The following image shows the menu that appears after you select Add Tab.



3. Select where you want to add the tab (before, after, or end).

   Each tab is one replace function. The new replace function is named INPUT#, however, you can change this to whatever name you want. In this way, you can add as many Input Lookup replace functions as you require.

For example, the following image shows a replace function (tab) named "Mammals." The lower pane of the Properties dialog box has columns for Match and Replace for various mammals, for example, cow to bovine.



Procedure: How to Remove a Replace Function

To remove a replace function (tab):

1. Right-click the tab of the replace function.
2. Select *Remove Tab*.

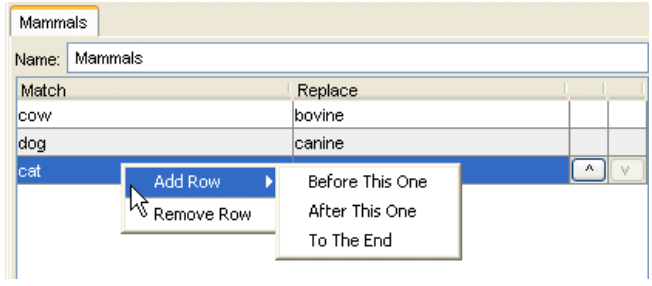
Procedure: How to Add a Match-Replace Pairing

Replace functions are made up of match-replace pairings. In the previous procedure, there are three static match-replace pairings. You define a match-replace pairing by typing in the values you desire.

To add and define a new match-replace pairing for a replace function:

1. Right-click an existing match-replace pairing.

The following image shows the menus that appear when you click an existing match-replace pairing.



2. Select *Add Row*.
3. Select where you want to add it (before, after, or end).

   Each row is one match-replace pairing. The new row is blank. You can add your match and replace values as you wish.

Procedure: How to Remove a Match-Replace Pairing

To remove a match-replace pairing (row):

1. Right-click the match-replace pairing.
2. Select *Remove Row*.

Using JDBC Replace Functions

JDBC replace functions are replace functions in which the match-replace values are taken from columns in a specified database table or from the results of an SQL statement run on the specified database. For example, you can tell iWay Transformer to use a database customer information table, reading the value from the "Full Name" column and replacing it with the value from the "Nickname" column. Unlike regular replace functions where the match-replace pairing is static, JDBC replace functions read the data from the specified columns of the next database row each time the transformation loops past the JDBC replace function while producing your output file.

Replace functions work like custom functions in that you must first define the function and then, apply it in the output item mapping value definition that you want to affect. In the case of replace functions, do this through the @REPLACE function. For more information, see *How to Define a JDBC Replace Function* on page 6-13.

For more advanced database functionality, such as matching across multiple input fields or running complex SQL queries, see *Using the @JDBCLOOKUP Function* on page 6-25.

The process of defining a replace function accounts for setting up the values, but not the mapping. You must map replace functions using the Mapping Builder.

JDBC Replace Functions as Input Data Sources

Since a JDBC replace function can be defined and used no matter what the input data format is, defining and using JDBC replace functions accesses multiple data input sources. For example, if your input data format is XML, you may be reading your input data from an XML file. By defining and using a JDBC replace function, you can also read input data from a database table. In fact, if you define many JDBC replace functions, you can use as many database sources as you like.

Procedure: How to Define a JDBC Replace Function

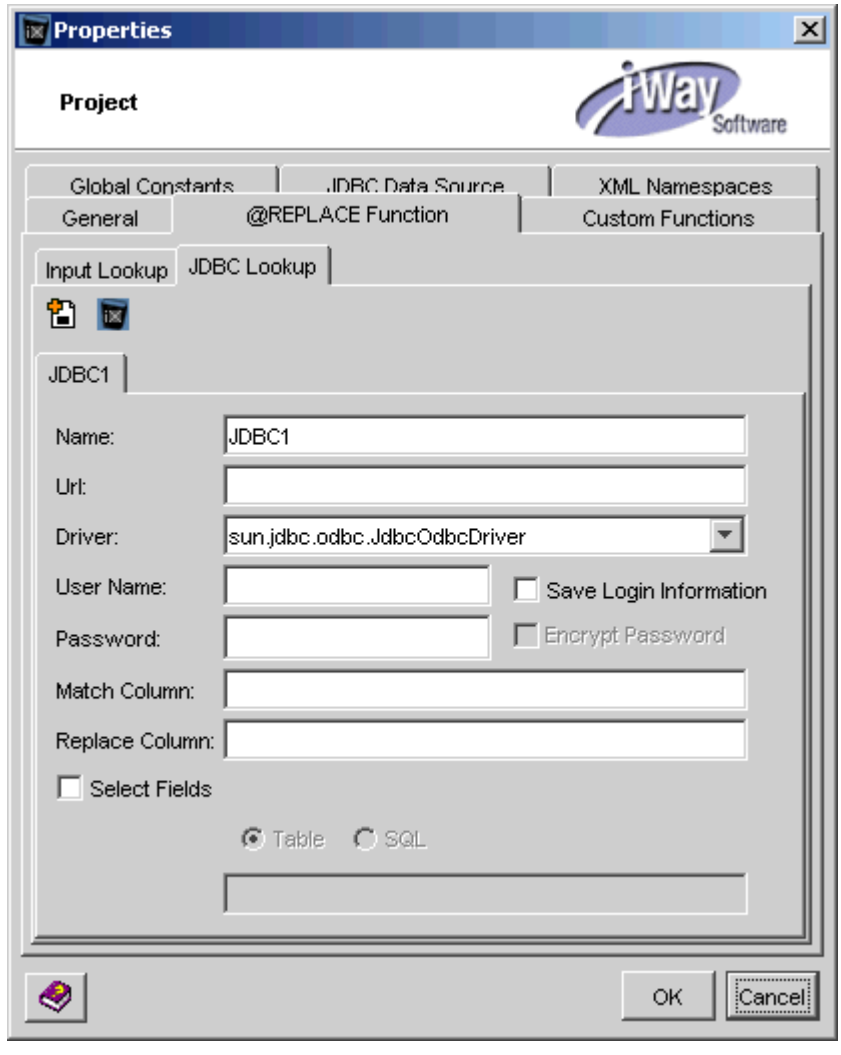
To define a JDBC replace function in iWay Transformer:

1. From the menu, select *Project* and then, *Properties*.

   The Project Properties dialog box opens with the General tab showing.

2. To define JDBC replace functions, click the *@REPLACE Function* tab and then, the *JDBC Lookup* tab.

The following image shows the Project Properties dialog box with the JDBC Lookup tab active.



A JDBC replace function is represented by one tab within the JDBC Lookup tab. To access the database you want, you complete the information for your JDBC replace function.

a. In the Match Column, type the name of the source Match Column.

b. In the target Replace Column, specify either a database table name or an SQL statement to run on the database.

If you specify an SQL statement, the match and replace values are taken from the result of the SQL statement.

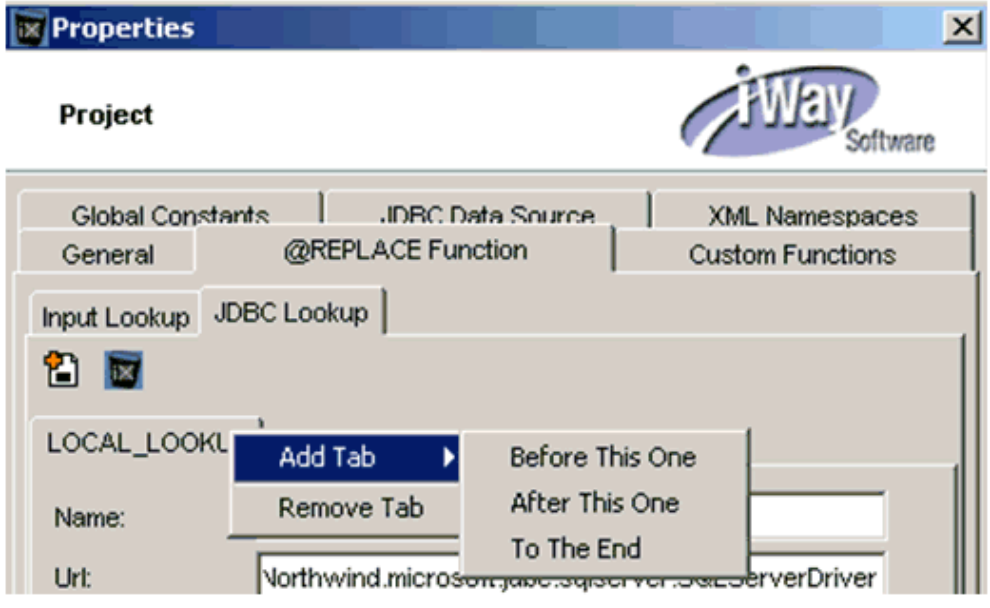
By default, there is one initial JDBC replace function named JDBC1 on the JDBC Lookup tab. You can change the name to whatever you want.

Procedure: How to Add a JDBC Replace Function

A JDBC replace function is represented by one tab within the JDBC Lookup tab. To add and define a new replace function (tab):

1. Right-click the tab of an existing JDBC replace function.

   The following image shows the menus that appear after you right-click the tab.



2. Select *Add Tab*.
3. Select where you want to add it (before, after, or end).

   Each tab is one replace function. The new replace function is named JDBC#. You can change this name to whatever you want.

Procedure: How to Remove a JDBC Replace Function

To remove a JDBC replace function (tab):

1. Right-click the tab of the replace function.
2. Select *Remove Tab*.

Using the Mapping Builder

The Mapping Builder is a simple tool for building a function for an output item value. You can access pre-defined functions and custom functions that you defined.

The function categories include:

- All Functions
- EDI Functions
- Numerical Functions
- Processing Functions
- Runtime Functions
- SWIFT Functions
- Security Functions
- String Functions
- Time Functions

Setting Function Parameters

After you select a working function, you can set the function parameters. Every function has associated parameters. The number of parameters required for your working function are shown in the right pane of the Mapping Builder. For example, the @CONCAT function in the in the following image takes three parameters.

If you do not know the parameters to specify for your working function, learn about your function in the Pre-defined Functions List. If your working function is a custom function, and you do not know the parameters to specify, consult the author of the custom function.

Function parameters can take four types of values:

- Input item
- Nested function
- Constant value
- Expression

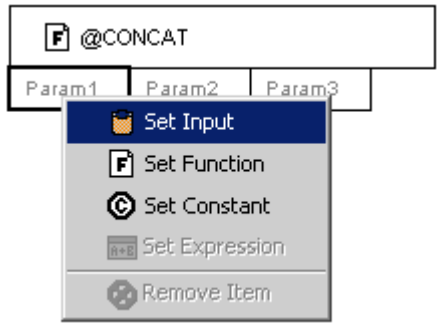
A function parameter can take the value of a specified input item. For information on how to set an input item, see *How to Set an Input Item* on page 6-21.

You can nest functions within functions. You specify another function as a parameter for your working function. For information on how to nest functions, see *How to Nest Functions* on page 6-22.

You can set a constant value for a function parameter. For information on how to set a constant value, see *How to Set a Constant Value for a Function* on page 6-23.

You can set an expression for a function parameter. For information on how to set an expression, see *How to Set an Expression for a Function* on page 6-24.

The following image shows a menu where you can select to set each of the previously discussed values.

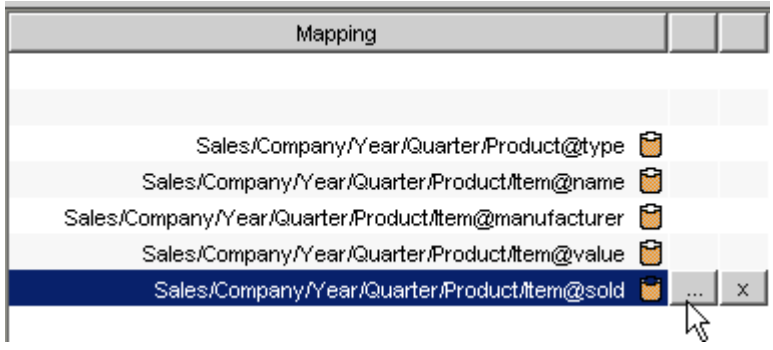


Pre-defined Functions

The Mapping Builder enables you to use functions to set output item values. You can use both custom functions and an extensive list of pre-defined functions. When specifying a function through the Mapping Builder, you must know what the specific function does and the parameters to specify for it.

Procedure: How to Open the Mapping Builder

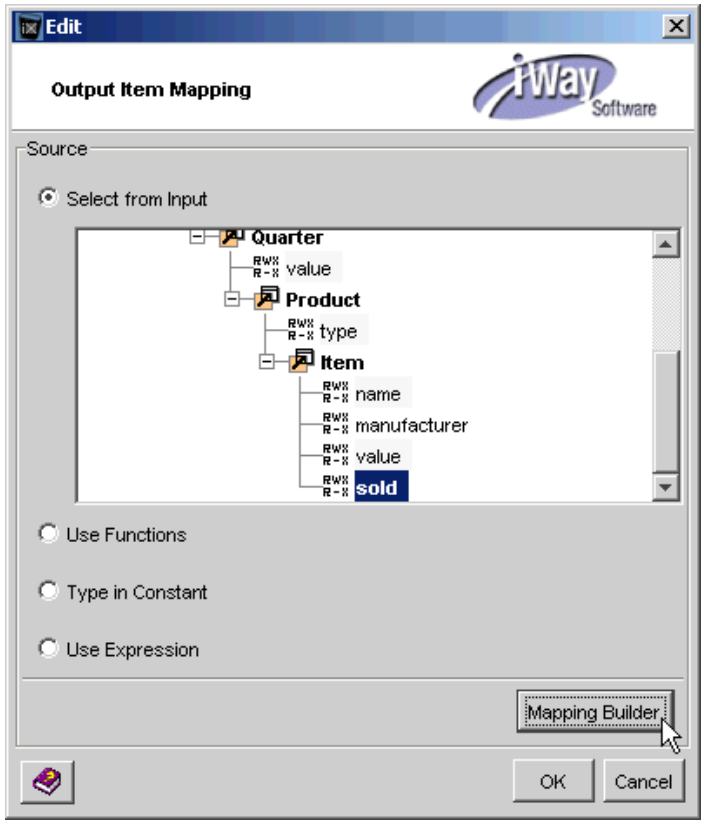
The following image shows the Mapping column.



To open the Mapping Builder:

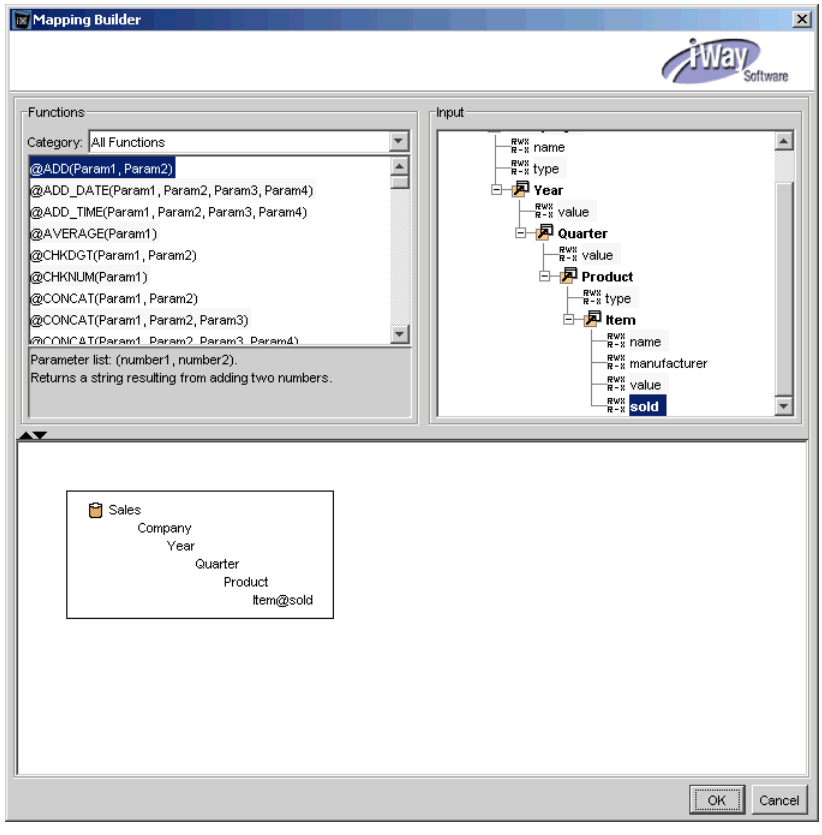
1. In the Mapping column, select an output item.
2. Click the ellipsis button.

The Edit dialog box opens as shown in the following image.



3. Click the *Mapping Builder* button.

The Mapping Builder opens as shown in the following image.



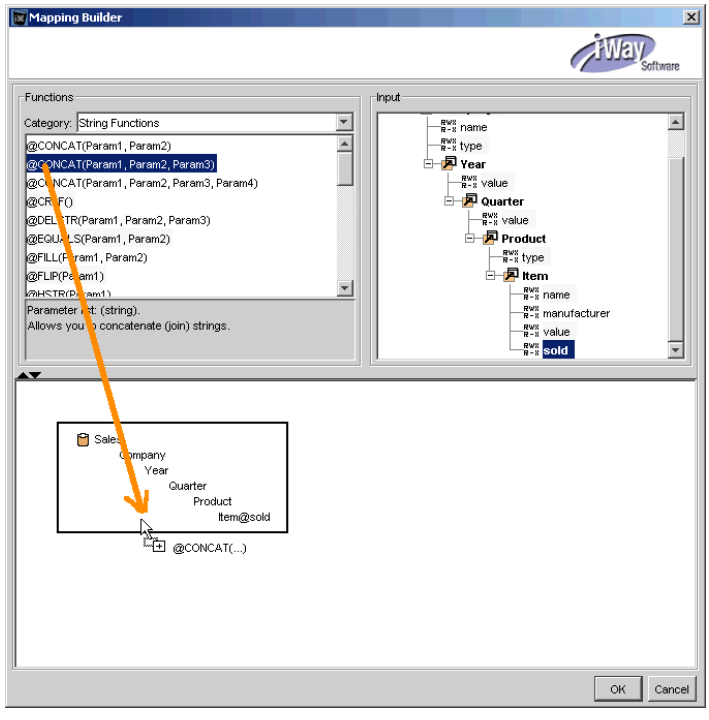
Procedure: How to Select a Function

To select a function:

1. From the Functions list in the top left pane of the Mapping Builder, select the category of function that you want to use.

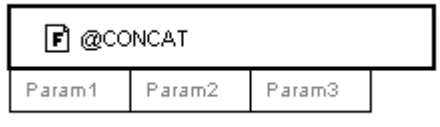
   For a list of function categories, see *Using the Mapping Builder* on page 6-16.

The following image shows String Functions in the Category drop-down list and a list of functions in the upper left pane of the Mapping Builder.



2. Select the specific function you wish to use and drag it onto the box in the bottom pane of the Mapping Builder.

   The function you choose becomes your working function.



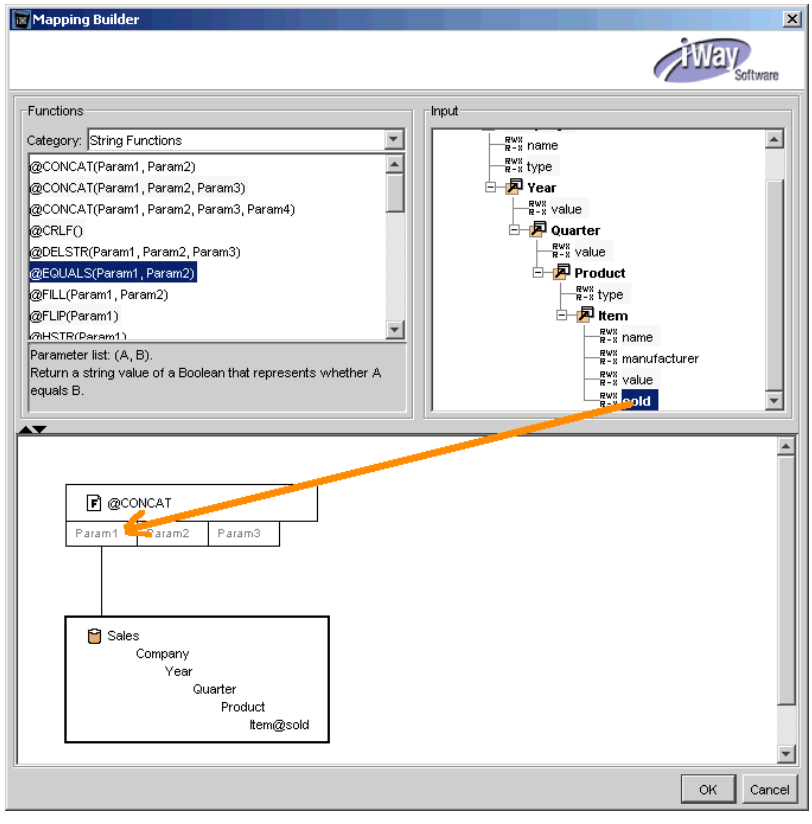
3. To change the working function, drag a new function onto the working function.

Procedure: How to Set an Input Item

To set an input item as the value for a function parameter:

1. Select the input item you wish in the top-right pane of the Mapping Builder as shown in the following image.

The following image shows String Functions in the Category drop-down list and a list of functions in the upper left pane of the Mapping Builder.



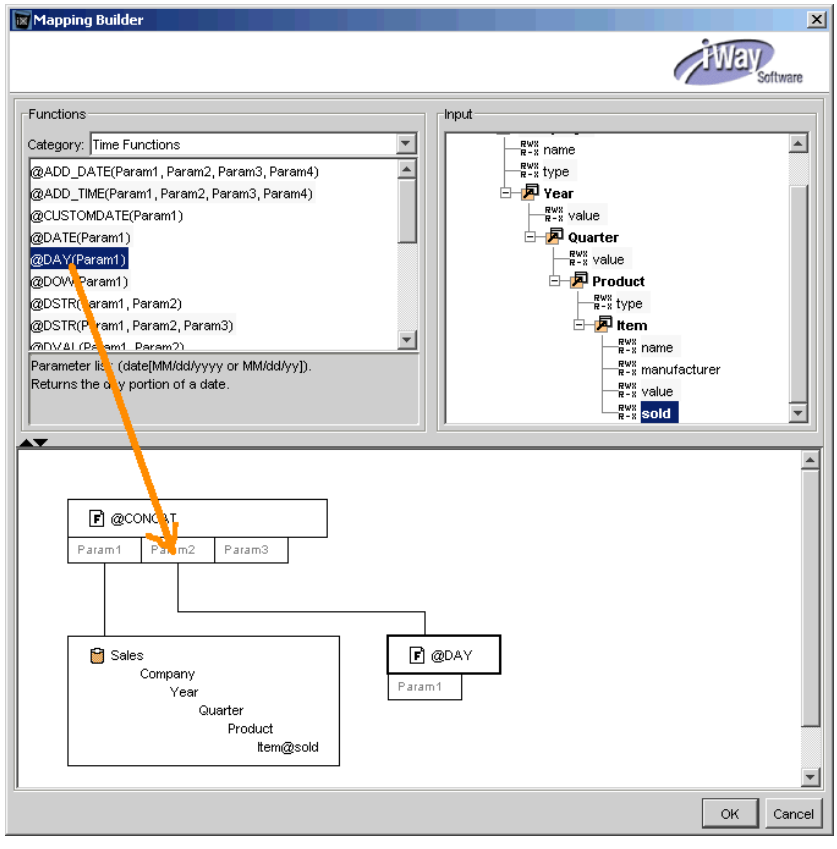
2. Drag the input item onto the appropriate parameter in the bottom pane.

Procedure: How to Nest Functions

To nest functions:

1. To specify another function as a parameter for your working function, select the function you want from the top-left pane of the Mapping Builder.

The following image shows Time Functions in the Category drop-down list and a list of functions in the upper left pane of the Mapping Builder.



2. Drag the function onto the appropriate parameter in the bottom pane.

   You must set the parameters for the nested function as well.

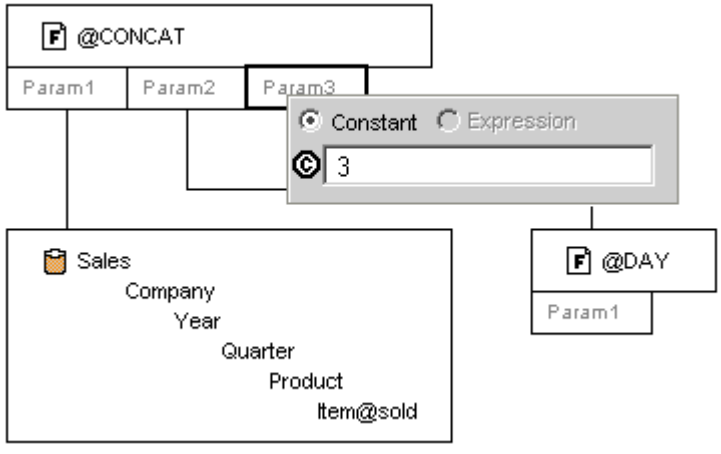
   Note: When nesting functions, you can only build the expression by adding it to the bottom only. Inserting is currently not supported.

Procedure: How to Set a Constant Value for a Function

To set a constant value:

1. Right-click the appropriate parameter box in the right pane of the Mapping Builder and select *Set Constant* from the context menu.

The following image shows the Constant option selected and a field where you can enter a value.



2. Type the value for your constant and press *Enter*.

Procedure: How to Set an Expression for a Function

To set an expression:

1. Right-click the appropriate parameter box in the right pane of the Mapping Builder and select *Set Expression*.
2. Type the value for your expression and press *Enter*.

   An expression is a concatenation of any number of input items and constant values in any order, in the following format:

   ```
   <input item>+'constant'+...
   ```

   For example:

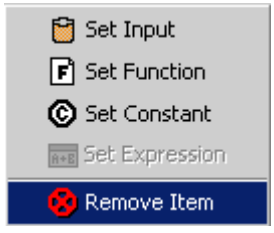
   ```
   'For '+tree/fruit+tree/fruit/year+': '+tree/fruit/apple/yield+
   ' bushels, '+tree/fruit/apple/avgweight+' grams avg.'
   ```

Procedure: How to Remove a Function Parameter

To remove a value for a function parameter:

1. Right-click the parameter in the bottom pane of the Mapping Builder.

A context menu opens as shown in the following image.



2. Select *Remove Item*.

Using the @JDBCLOOKUP Function

The @JDBCLOOKUP(Param1, Param2) function returns a matched value from a database using an SQL statement. The SQL statement can be dynamic based on input from other transform functions. If more than one value is matched, then the last value is returned.

When no result is found in the database, the function returns an empty string. When more than one result found, the function returns the last result. Regardless of the number of columns in the result set, the first column is always taken. During a transformation, if the SQL statements are the same, then the result will be the same.

Parameters:

Param1: String that represents a globally defined JDBC connection configuration.

Param2: SQL statement that can be created using SQL Builder.

Example:

```
@JDBCLOOKUP('LOOKUP_TEST', {'SELECT field1 FROM LOOKUP_TABLE WHERE field2
' = '+@QUOTE(Customer/Person/Name)})
```

where

`LOOKUP_TEST`

Is the name of a JDBC connection configuration.

```
{'SELECT field1 FROM LOOKUP_TABLE WHERE field2 ' =
'+@QUOTE(Customer/Person/Name)}
```

Is an SQL statement.

JDBC Connection Pooling

JDBC connection pooling is provided. Currently, the default max pool size is set to 10 connections. To modify this value, locate the JDBCDataSource_en_US.properties file and update the following property:

```
transform.jdbcDataSource.maxConnectionPoolSize=10
```

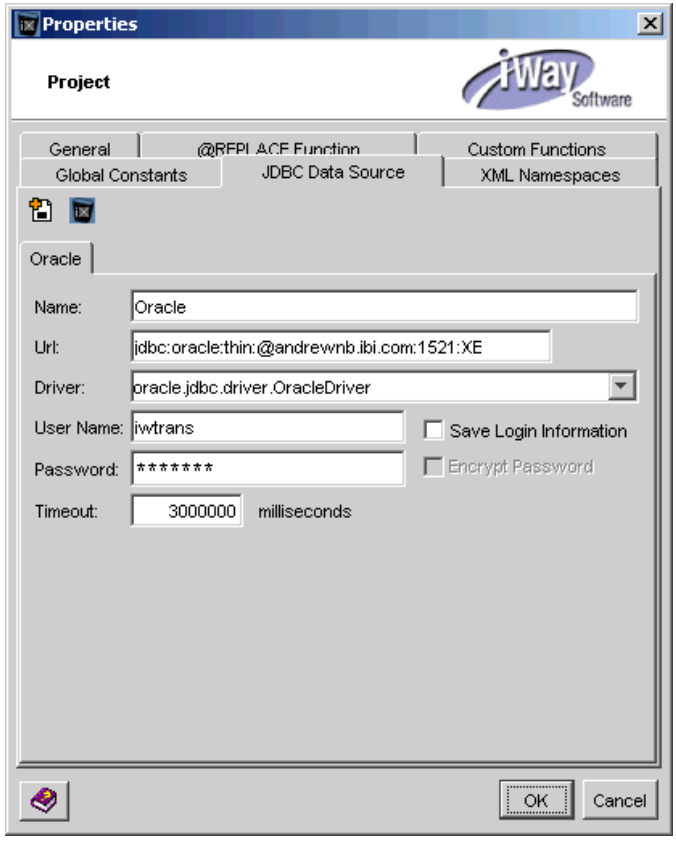
Procedure: How to Create an SQL Statement Using SQL Builder

To create an SQL statement using SQL Builder that can be used for the @JDBCLOOKUP(Param1, Param2) function:

1. From the menu bar, click *Project* and select *Properties*.

   The Properties dialog box opens.

2. Click the *JDBC Data Source* tab.



3. Click the *Add tab* icon  to define a new JDBC connection.

4. Perform the following steps:

   a. Provide a name for the JDBC connection.

   This name is used as a string value for the Param1 parameter in the @JDBCLOOKUP function.

b. Type the Url to the database.

c. Select a JDBC driver from the drop-down list.

d. Type a valid user name and password to connect to the database.

e. Specify a timeout value in milliseconds.

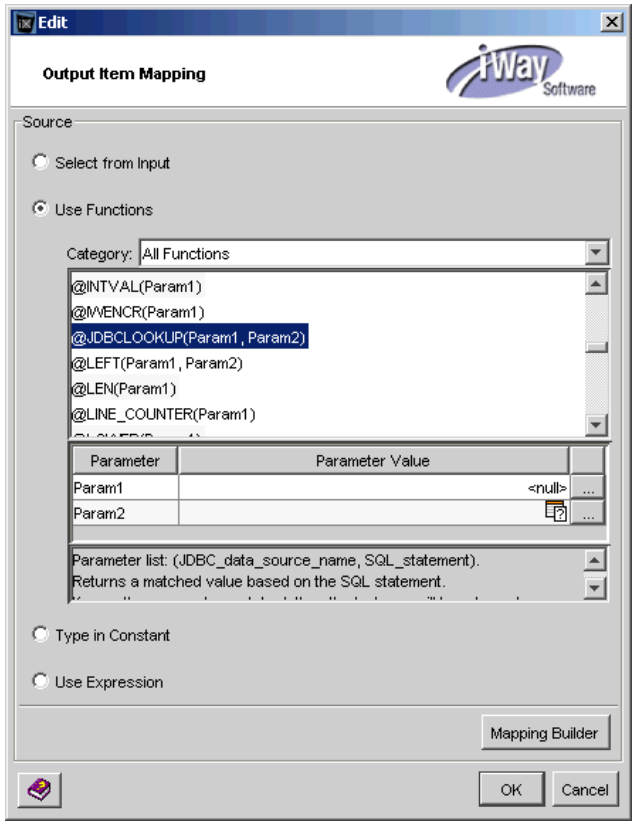
Note: If the timeout value is omitted, or the value is 0, then timeout is not enabled.

5. Click *OK*.

6. In the Mapping column, select an output item.

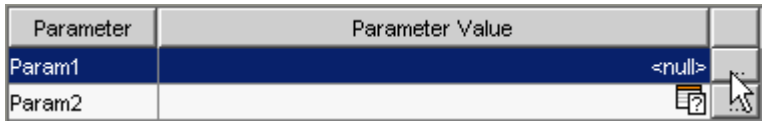
7. Click the ellipsis button.

The Edit dialog box opens as shown in the following image.



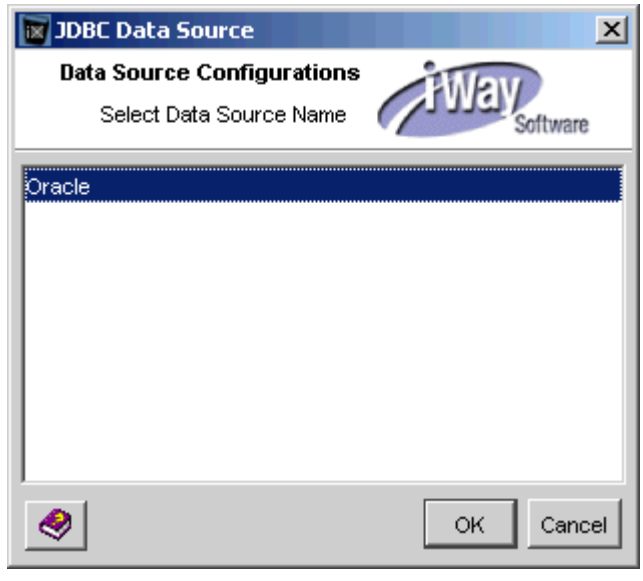
8. Click the *Use Functions* radio button.

9. Select the @JDBCLOOKUP(Param1, Param2) function from the list.



10. Click the ellipsis button for Param1.

The JDBC Data Source dialog box opens.

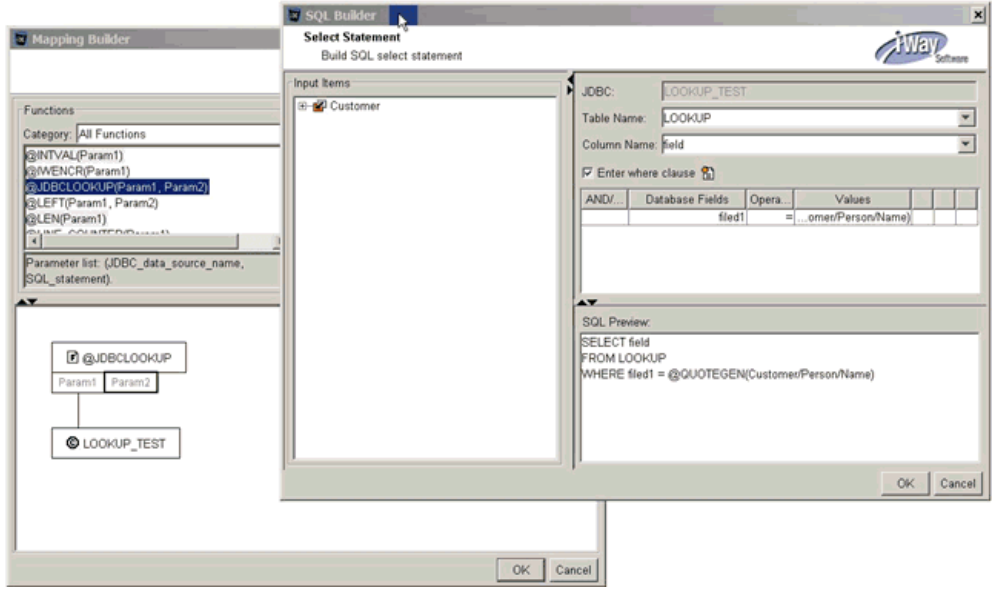


11. Select the name of a defined JDBC connection, for example Oracle, and click *OK*.

You are returned to the Edit dialog box.

12. Click the *Mapping Builder* button.

The Mapping Builder opens.

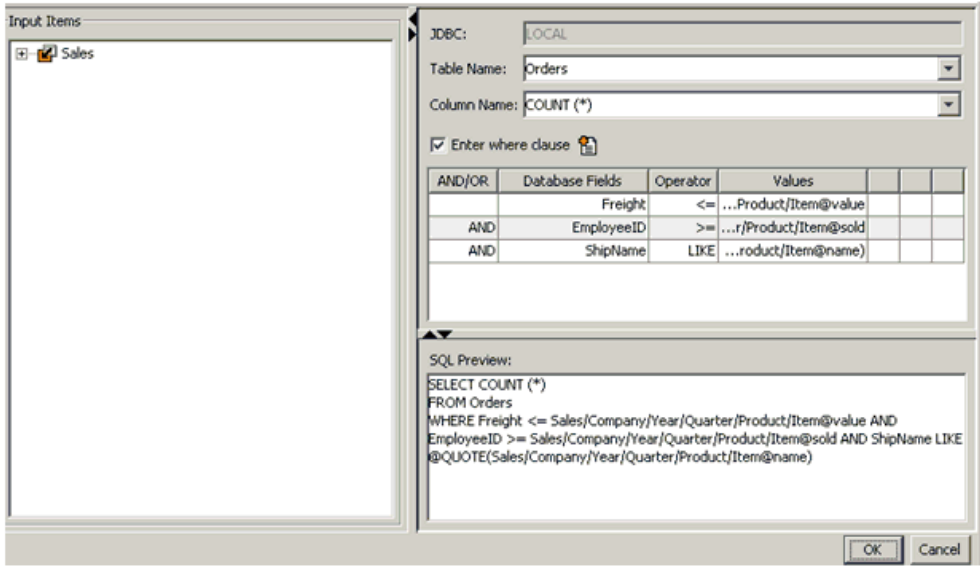


13. Double-click *Param2* to open the SQL Builder.

Since the SQL statement is an expression, the Where clause can contain the following:

- Constant
- Function
- Input Context
- Unlimited AND & OR

Note: In the Column Name field, you can also type * or Count (*) to return more advanced datasets.



14. Click *OK* once you have finished creating your SQL statement.

APPENDIX A

Pre-Defined Functions

Topics:

- EDI Functions
- Numeric Functions
- Processing Functions
- Runtime Functions
- SWIFT Functions
- Security Functions
- String Functions
- Time Functions

A function is Java code that performs a calculation upon, or manipulation of, input data. You can apply functions to produce the output value within its output item mapping dialog.

iWay Transformer provides more than 60 pre-defined functions. This section provides detailed information about these functions.

EDI Functions

The following table lists and describes the EDI functions that are available in iWay Transformer.

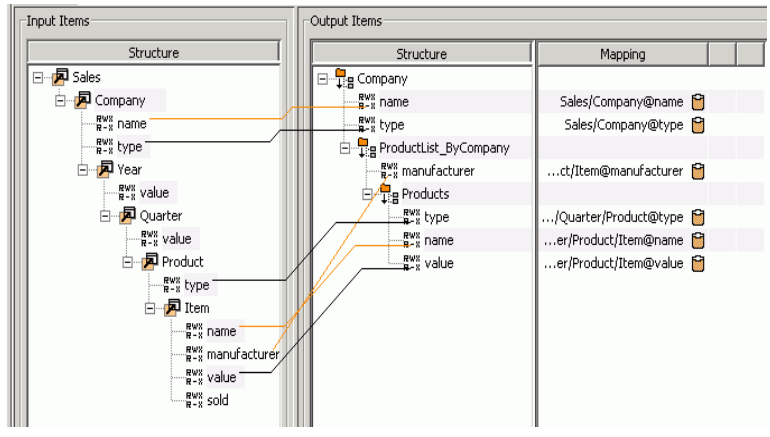
EDI Function	Description
@COUNT_GROUP()	Returns occurrences of a group within an EDI message. **Parameters:** None.
@COUNT_SEGMENT()	Returns occurrences of a segment within a transaction set. **Parameters:** None.
@COUNT_SEGMENT (Param1)	Returns occurrences of the specified segment within a transaction set. **Parameter:** Param1: Name of the segment and must be a constant.
@COUNT_TRANSACTION()	Returns occurrences of the transaction within a group. **Parameters:** None.
@LINE_COUNTER (Param1)	Returns occurrences of the specified segment within the same loop instance. The counter is reset when a new loop starts. **Parameter:** Param1: Name of the segment and must be a constant. It is based on the LN segment description.
@PARENT_SEGMENT_ID()	Returns the hierarchical level of the segment's parent based on the HL segment description. **Parameters:** None.
@SEGMENT_ID_SEQUENCE()	Returns the hierarchical level of the segment based on the HL segment description. This function should be used once per segment. **Parameters:** None.

Numeric Functions

The following table lists and describes the numerical functions that are available in iWay Transformer.

Numeric Function	Description
@ADD (Param1, Param2)	Returns the result of adding two numbers specified by parameters. **Parameters:** Param1: Number to add to Param2. Param2: Number to add to Param1. **Example:** @ADD(24.01, 12.02) returns 36.03.
@AVERAGE (Param1)	Returns the average of the specified parameter values within the same parent instance. This function should be used in combination with Agg looping. **Parameter:** Param1: Number that represents the value to average. Usually, it's mapped to the input document. **Note:** The looping settings of the parent and grandparent of the attribute or element that uses the @AVERAGE function must be carefully set. For more information, see *Parent Properties* on page 5-25.

Numeric Function	Description

@AVERAGE (*continued*)	**Example:**
	In the following sample image, output element item ValueAverage has the value @AVERAGE(Sales/Company/Year/Quarter/Product/Item@value).
	 
	
	where the output obtained is:
	`<Sales_Totals>`
	`<companyName>Video and Sound Card Express</companyName>`
	`<Statistics>`
	`<itemValueAverage>139.7407</itemValueAverage>`
	`<allTimeSales>95022.02</allTimeSales>`
	`</Statistics>`
	`</Sales_Totals>`
	In this XML to XML transformation example, the Sales_Totals output parent has looping set to False and the Statistics parent has looping set to Agg. The desired output is also obtained with Sales_Totals looping set to Agg, but not with True.

Numeric Function	Description
@CHKNUM (Param1)	Returns true string if the specified parameter is a valid number. Use this function to ascertain if the parameter is a number (whole or decimal). Returns true or false. **Parameter:** Param1: Number to validate. **Examples:** @CHKNUM('1.1') returns true. @CHKNUM('abcd') or @CHKNUM('1,234.55') returns false.
@COUNT	Counts input items processed and returns the next value sequentially. The first loop through @COUNT initializes the counter to 1. Subsequent passes increment the count to the next number. **Parameters:** None.
@DIVIDE (Param1, Param2)	Returns the result of dividing two numbers specified by parameters. **Parameters:** Param1: Number that represents the dividend. Param2: Number that represents the divisor. **Example:** @DIVIDE('24.02', '12.01') returns 2.0.
@INT (Param1)	Returns the first occurrence of the integer value found in the specified parameter. If the integer cannot be found, the number 0 (zero) is returned. **Parameter:** Param1: String to be checked. **Examples:** @INT('45.60 or 65.60') returns 45. @INT('in the summer of 1998 and 1999') returns 1998. @INT('time is 23:11:56') returns 23. @INT('last year') returns 0.

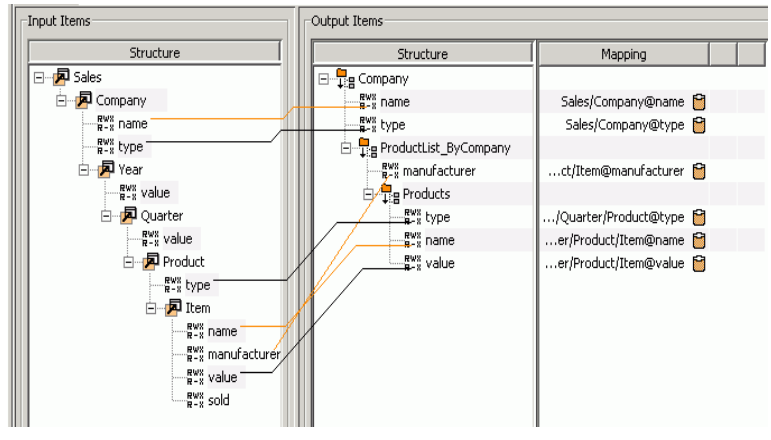
Numeric Function	Description
@INTVAL (Param1)	Returns the specified parameter value if it is a valid integer. Otherwise, it returns the number 0 (zero). **Parameter:** Param1: String to be checked. **Examples:** @INTVAL('45') returns 45. @INTVAL('45.12') or @INTVAL('dollars 1.23') returns 0.
@MULTIPLY (Param1, Param2)	Multiplies two numbers to return a string. **Parameters:** Param1: Number to multiply by Param2. Param2: Number to multiply by Param1. **Example:** @MULT (12.01, 2.00) returns 24.02.
@NUM (Param1)	Converts an integer string to a numeric value. Use this function to ascertain if a parameter is an integer (whole number). The function returns the parameter or an error. **Parameter:** Param1: Numeric string to be converted to an integer value. **Example:** @NUM('45.12') returns 45.12.
@NUM_CHR (Param1)	Returns the ASCII character that corresponds to a number. **Parameter:** Param1: Integer number between 1 and 127 to represent an ASCII number. **Example:** @NUM_CHR ('66') returns 'B'.

Numeric Function	Description
@RANDOM (Param1)	Returns a pseudo random number. **Parameter:** Param1: Number used to seed the random number generator. If Param1 = -1, the seed is initialized randomly. If Param1 = *n* (where *n* is not 0 or -1), the seed is initialized to *n*. If Param1 = 0, the next random number is generated based on the existing seed. After initializing the seed, continue with @RANDOM ('0') to generate the next random number.
@RANGE (Param1, Param2, Param3)	Checks if a number falls within a range and returns a Boolean true or false. **Parameters:** Param1: Value is checked. Param2: Value that represents the lower limit of the range. Param3: Value that represents the upper limit of the range. **Note:** The range is given by Param2 to Param3, inclusive. **Example:** @RANGE('10', '5', '15') evaluates to TRUE.
@ROUND (Param1,Param2, Param3)	Extracts a specified part of a number and rounds the result to an integer. **Parameters:** Param1: Number subjected to the operation. Param2: Number of digits to be extracted from the integer part of Param1. Digits are counted from the left of the decimal separator. Param3: Number of digits to be extracted from the decimal part of Param1. Digits are counted from the right of the decimal separator. **Example:** @ROUND(345.995, 2, 2) returns 46.00.

Numeric Function	Description
@SCALE_ROUND (Param1, Param2, Param3)	Returns the specified parameter rounded by the specified scaling type. Enables you to scale a number and specify the way it is rounded during scaling. **Parameters:** Param1: Number subjected to the operation. Param2: Number of digits to the right of the decimal place keep. The value must be a non-negative integer. Param3: Type of rounding required. The following are possible values: ROUND_CEILING: Round towards positive infinity. ROUND_DOWN: Round towards zero. ROUND_FLOOR: Round towards negative infinity. ROUND_HALF_DOWN: Round towards nearest number. If equidistant to two numbers, round down. ROUND_HALF_EVEN: Round towards nearest number. If equidistant to two numbers, round towards even number. ROUND_HALF_UP: Round towards nearest number. If equidistant to two numbers, round up. ROUND_UNNECESSARY: No rounding is necessary. Use this rounding mode if you want to make sure that the output value has the exact number of decimals specified in Param2. If the number of decimals is different, an error is returned. ROUND_UP: Round away from zero. **Example:** @SCALE_ROUND(5.111, 2, ROUND_UP) returns 5.12.

Numeric Function	Description
@STR (Param1, Param2)	Converts a number to an alpha string according to a picture mask. Use this function to format numbers. **Parameters:** Param1: Number to be converted into an alpha string. Param2: Picture mask format for the string. For more information, see *Numeric Pictures* on page A-33. **Examples:** @STR('45.12', '##.#') returns '45.1'. @STR('75','#.00') returns 75.00. @STR('567.1', '#.00') returns 567.10. **Note:** Since a picture mask is used, the function may cause rounding, for example, @STR(39.999,'#.00') returns '40.00'.
@SUBTRACT (Param1, Param2)	Subtracts two numbers, returning the remainder as a string. **Parameters:** Param1: Float number from which to subtract Param2. Param2: Float number to subtract from Param1. **Example:** @SUBTRACT(24.02, 11.01) returns 13.01.

Numeric Function	Description

@SUM
 	
(Param1)
	Returns the sum of the numerical values of a specified input item.
	
	Parameter: Param1: Input item that has a numeric value.
	
	Note: The looping settings of the parent and grandparent of the attribute or element that uses the @AVERAGE function must be carefully set. For more information, see *Parent Properties* on page 5-25.
	
	Example: In the following sample image, output element allTimeSales has value @SUM(@MULTIPLY(Sales/Company/Year/Quarter/Product/Item @value, Sales/Company/Year/Quarter/Product/Item@sold)).
	
	 
	
	Where the output obtained is:
	
	`<Sales_Totals>`
	`<companyName>Video and Sound Card Express</companyName>`
	`<Statistics>`
	`<itemValueAverage>139.7407</itemValueAverage>`
	`<allTimeSales>95022.02</allTimeSales>`
	`</Statistics>`
	`</Sales_Totals>`
	
	In this XML to XML transformation example, the Sales_Totals output parent has looping set to False, and the Statistics parent has looping set to Agg. The desired output is also obtained with Sales_Totals looping set to Agg, but not with True.

Numeric Function	Description
@VAL (Param1, Param2)	Returns the number that matches the specified picture mask. This function returns a value only if the picture mask matches the input format. For example, if the picture mask is # and the input is 7.3, the function returns an error, not 7 as might be expected. Use this function to retrieve the numeric value from a string. Do not use this function to format a number. To format a number, use the @STR function. **Parameters:** Param1: String that contains the number to be retrieved. Param2: Format in which the number is stored in the string; must be specified as a constant in the Mapping Builder. For more information, see *Numeric Pictures* on page A-33. **Examples:** @VAL('30.11 dollars plus a fee of 40 dollars','##') returns 40. @VAL('30.11 dollars plus a fee of 40 dollars','##.##') returns 30.11. **Note:** If you need to format a number with commas such as <X>123,456,789</X>, you can remove commas by using the @CONCAT and @SUBSTR functions. For example: @CONCAT(@SUBSTR(X,'1','3'),@SUBSTR(X,'5','3'),@SUBSTR(X,'9','3'))

Processing Functions

The following table lists and describes the processing functions that are available in iWay Transformer.

Processing Function	Description
@EDIT (Param1, Param2, Param3)	Returns a value formatted using the specified picture mask and formatter class name. For more information, see *Numeric Pictures* on page A-33. **Parameters:** Param1: String representing the formatter class name. Current implementation has support only for the DecimalFormat class. Param2: String representing the picture mask to be applied to Param3. Param3: Number to be formatted. **Examples:** @EDIT('DecimalFormat','#,###.##','123456.78) returns 123,456.78. @EDIT('DecimalFormat','000','12') returns 012.
@GETCONSTANT (Param1)	Returns the value of the global constant specified by the parameter. **Parameter:** Param1: Global constant defined in the Global Constant section of the Project Properties pane. **Example:** Assume you defined the following global constant in project properties: Name: COMPANY_ADDRESS Value: 1234 1st Avenue. @GETCONSTANT('COMPANY_ADDRESS') returns "1234 1st Avenue".

Processing Function	Description
@IF (Param1,Param2, Param3,Param4, Param5)	Allows for conditional selection. If the condition is true, it returns the True option. Otherwise, it returns the False option. **Parameters:** Param1: Left operand. Param2: Conditional operator; the possible arguments are: • = = equal to • != not equal to • >= equal to or greater than • <= equal to or less than • > greater than • < less than Param3: Right operand. Param4: (true_option). The string is returned if the condition is true. Param5: (false_option). The string is returned if the condition is false. **Example:** The variable X has a value that fluctuates between 4 and 8. @IF(X<'10', 'ABC', 'DEF') always returns 'ABC'.
@INPUT_CONTENT	Returns the contents of the entire input data file as a string. **Parameters:** None.

Processing Function	Description

@JDBCLOOKUP
(Param1, Param2)
	Returns a matched value from a database using an SQL statement. The SQL statement can be dynamic based on input from other transform functions. If more than one value is matched, then the last value is returned.
	Parameters:
	Param1: String that represents a globally defined JDBC connection configuration.
	Param2: SQL statement that can be created using SQL Builder.
	Example:
	`@JDBCLOOKUP('LOOKUP_TEST', {'SELECT field1 FROM LOOKUP_TABLE WHERE field2 ' = '+@QUOTE(Customer/Person/Name)})`
	where
	`LOOKUP_TEST`
	Is the name of a JDBC connection configuration.
	`{'SELECT field1 FROM LOOKUP_TABLE WHERE field2 ' = '+@QUOTE(Customer/Person/Name)}`
	Is an SQL statement.

@NULL	Returns a null output, that is, no output. Useful, for example, as either the true_option or false_option in an @IF function. **Parameters:** None.
@REPLACE (Param1, Param2)	Calls defined replace functions and replaces every matched string. If the match is not found, the first parameter is returned. For more information, see *Defining Replace Functions* on page 6-8. **Parameters:** Param1: Input item into which to make changes. Param2: Replace function name as defined. **Example:** @REPLACE('11009333009', 'REPLACE_009_WITH_Add') returns 11Add333Add.

Processing Function	Description
@SIMPLE_REPLACE (Param1, Param2)	Calls defined replace functions and replaces first matched string (exact match). If the match is not found, the first parameter is returned. **Parameters:** Param1: Input item into which to make changes. Param2: Replace function name as defined. **Example:** @SIMPLE_REPLACE('009', 'REPLACE_009_WITH_Add') returns Add.

Runtime Functions

The following table lists and describes the runtime functions that are available in iWay Transformer.

Runtime Function	Description
@IWENCR (Param1)	Encrypts the value using the iWay Service Manager internal encryptor. **Parameter:** Param1: String to be encrypted. **Example:** @IWENCR('1234') returns the value in a format similar to: @ENCR(32533101323532123122319631833137).
@SREG (Param1, Param2)	Returns the iWay Service Manager special register defined by the specified parameter. Executed if transform is run on the server. **Parameters:** Param1: String that represents the name of the special register to be returned. Param2: String that represents the default value to return if the special register identified by Param1 is not found.

Runtime Function	Description
@SET_SREG (Param1, Param2, Param3, Param4)	Sets iWay Service Manager special register to specified value and returns Param4. Executed if transform is run on the server. The register name is specified in Param1, the new value is specified in Param2, the type in Param3, and Param4 is returned upon successful execution. **Parameters:** Param1: String that represents the name of the special register. Param2: String that represents the new value of the special register. Param3: Number that represents type of the special register. Possible values include: • 2: user-defined variable • 3: user-defined emit header Param4: String that represents the expected return value of this function. **Example:** @SET_SREG ('custom_functions_location', 'tools/transformer/custom_functions','2','custom functions location is set') returns custom function location is set.
@SREG_EXISTS (Param1)	Checks if an iWay Service Manager special register is already defined and returns a true or false response. Executed if transform is run on the server. **Parameters:** Param1: String that represents the name of the special register. **Example:** @SREG_EXISTS('custom_functions_location') Returns true if the register was already defined. Returns false if the register was not defined.

Runtime Function	Description
@REMOVE_SREG (Param1, Param2)	Removes iWay Service Manager special register and returns Param2. Executed if transform is run on the server. **Parameters:** Param1: String that represents the name of the special register. Param2: String that represents the expected return value of this function. **Example:** @REMOVE_SREG ('custom_functions_location','custom functions location is removed') Returns custom function location is removed.

SWIFT Functions

The following table lists and describes the SWIFT functions that are available in iWay Transformer. Within SWIFT document specifications, there are data segments that require special handling to parse them properly. These SWIFT functions enable users to parse such data segments.

SWIFT Function	Description
SWIFT_SEG_37A_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_38_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_61_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.

SWIFT Function	Description
SWIFT_SEG_68A_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_68B_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_68C_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_71B_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_90C_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_90D_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.
SWIFT_SEG_ZZ_ CUSTOM_PARSER (Param1, Param2)	**Parameters:** Param1: String that represents the field value. Param2: Number that represents the index of the element; must be a constant.

Security Functions

The following table lists and describes the security functions that are available in iWay Transformer.

Security Function	Description
@CHKDGT (Param1, Param2)	Checks that a specified alphanumeric character is a number or a letter and returns 'true' if it is a number or 'false' if it is a letter. **Parameters:** Param1: Alpha string that represents the number to be checked. Param2: Position of the character in Param1 to check. The position numbering starts at 0 (zero) from the left. Enter as a constant in the Mapping Builder. **Examples:** @CHKDGT('4abc', '0') returns true. @CHKDGT('6a89', '1') returns false.

String Functions

The following table lists and describes the string functions that are available in iWay Transformer.

String Function	Description
@CONCAT (Param1, Param2, Param3, Param4)	Returns the concatenations of the specified parameters. The following signatures are available: @CONCAT(Param1, Param2) @CONCAT(Param1, Param2, Param3) @CONCAT(Param1, Param2, Param3, Param4) **Parameters:** Param1: String to be concatenated. Param2: String to be concatenated. Param3: String to be concatenated. Param4: String to be concatenated. **Example:** @CONCAT('The cow ', 'jumped ', ' over ', ' the moon') returns 'The cow jumped over the moon'.
@CRLF	Returns Carriage Return and new Line Feed characters. **Parameters:** None. **Example:** @CONCAT('First line',@CRLF(),'Second line') returns First line Second line.

String Function	Description
@DELSTR (Param1, Param2, Param3)	Deletes characters from an alpha string. **Parameters:** Param1: Alpha string or alpha string expression. Param2: Position of the first character to be deleted. Param3: Number of characters to be deleted, beginning with Param2 and continuing to the right. **Examples:** @DELSTR('ABCD', '2', '1') deletes the second letter of the string and returns 'ACD'. @IF(Y<0, @DELSTR(X, '1', '1'), @IF(Y>0, @DELSTR(X, '2', '1'), X)). If X contains a character string of length greater than or equal to 2, the expression removes either the first or second character, or leaves the string intact, according to the value that appears in column Y (negative, positive, or zero).
@EQUALS (Param1, Param2)	Compares two strings and returns 'true' if they are equal or 'false' if they are not. **Parameters:** Param1: String. Param2: String. **Example:** @EQUALS('BA', 'AB') return 'false'.
@FILL (Param1, Param2)	Repeats an alpha string or expression 'n' times. **Parameters:** Param1: An alpha string or expression. Param2: The number of times the string is repeated. **Example:** @FILL('*',5) creates a string of five asterisks '*****'. **Note:** This function accepts a maximum of 32K.
@FLIP (Param1)	Reverses an alpha string, or the result of an alpha expression, to its mirror image. **Parameter:** Param1: Alpha string or alpha string expression. **Example:** @FLIP('Good') returns 'dooG'.

String Function	Description
@HSTR (Param1)	Returns the hexadecimal (base 16) string value of a decimal (base 10) number. **Parameter:** Param1: Decimal (base 10) number or numeric expression that represents a decimal number. **Example:** @HSTR('15') returns 'F'. @HSTR('16') returns '10'.
@HVAL (Param1)	Returns the decimal (base 10) value of a hexadecimal (base 16) number. **Parameter:** Param 1: Alpha string that represents a hexadecimal (base 16) number. **Example:** @HVAL('FF') returns 255. @HVAL('10') returns 16.
@INSERT (Param1, Param2, Param3, Param4)	Inserts one string into another. **Parameters:** Param 1: Alpha string that represents the target string. Param 2: Alpha string that represents the source string. Param 3: Number that represents the starting position in Param1. Param 4: Number that represents the number of characters from Param2 to be inserted into Param1. **Example:** @INSERT('abcde', 'xxx', '3', '2') returns 'abxxcd'.
@INSTR (Param1, Param2)	Returns a number that represents the first position of a sub-string within an alpha string or alpha expression. **Parameters:** Param1: Alpha string or alpha expression. Param2: Alpha string that will be the search argument in Param1. **Examples:** @INSTR('abcd', 'b') returns 2. @INSTR('ABCDEF', 'DE') returns 4. **Note:** If the search argument is not found, the function returns 0 (zero).

String Function	Description
@LEFT (Param1, Param2)	Returns a specified number of characters from an alpha string, starting from the left. **Parameters:** Param 1: Alpha string. Param2: Number of characters to be returned, starting from the left. **Example:** @LEFT('abcdefg', '3') returns 'abc'.
@LEN (Param1)	Returns the defined length of an alpha string. **Parameter:** Param1: Input alpha string. **Example:** @LEN ('abcdefg') returns 7. **Note:** The function right-justifies the string (for example, removes trailing blanks) before commencing.
@LOWER (Param1)	Converts a string to all lowercase. **Parameter:** Param 1: Alpha string. **Example:** @LOWER('Who said THAT?') returns 'who said that?'.
@LPAD (Param1, Param2)	Returns the specified parameter padded to the left using space characters as a padding string. The padding string is inserted to the left of the string until the length is reached. If padding is not performed (parameters are invalid or the length is smaller than the string length) the first parameter is returned. **Parameters:** Param1: String to be padded. Param 2: Number that represents the length of the returned string. **Examples:** Square brackets are not part of the data; they are used to visually mark the length of the padded string. @LPAD('constant','12') returns [constant]. @LPAD('constant','3') returns [constant].

String Function	Description
@LPAD (Param1, Param2, Param3)	Returns the specified parameter padded to the left using the specified padding string. The padding string is inserted to the left of the string until the length is reached. If padding is not performed (parameters are invalid or the length is smaller than the string length), the first parameter is returned. **Parameters:** Param1: String to be padded. Param2: Number that represents the length of the returned string. Param3: String used for padding. **Examples:** Square brackets are not part of the data; they are used to visually mark the length of the padded string. @LPAD('constant','12', 'L') returns [LLLLconstant]. @LPAD('constant','3', 'L') returns [constant].
@LTRIM (Param1)	Removes leading white space (blanks, tabs, line feeds) from an alpha string or an alpha expression. **Parameter:** Param1: Input alpha string. **Example:** @LTRIM(' John') returns 'John'.
@MID (Param1, Param2, Param3)	Extracts a specified number of characters (a sub-string) from an alpha string. **Note:** Will be deprecated in future releases. Use @SUBSTR. **Parameters:** Param 1: Input alpha string. Param2: Number that represents the starting position of the sub-string within Param1. Param3: Number of characters to be extracted (length of sub-string). **Example:** @MID('John', '3', '2') returns 'hn'.

String Function	Description
@NOT_EQUALS (Param1, Param2)	Compares two strings and returns 'false' if they are equal or 'true' if they are not. **Parameters:** Param1: String. Param2: String. **Example:** @EQUALS('BA', 'AB') return 'true'.
@QUOTE (Param1)	Returns the specified parameter delineated by single quotation marks. **Parameter:** Param1: String to be quoted. **Examples:** @QUOTE('quote me') returns 'quote me'. @QUOTE('23') returns '23'.
@QUOTEGEN (Param1)	Generates 'Where' SQL statements. Generates single quotation marks around a string if it is of type varchar, but not if it is of type integer or number. **Parameter:** Param1: String; can be an input item, a constant, or the result of another function. **Examples:** @QUOTEGEN('HELLO') returns 'HELLO'. @QUOTEGEN('1234') returns 1234. @CONCAT('SELECT column1 from table where column2 = ', @QUOTEGEN(parent/child/value)) returns: • SELECT column1 from table where column2 = 'value', IF parent/child/value type is varchar. • SELECT column1 from table where column2 = value, IF parent/child/value type is integer or number.

String Function	Description
@REP (Param1, Param2, Param3, Param4)	Replaces an alpha sub-string within a string with another sub-string. **Parameters:** Param1: Target alpha string or expression where the replacement will take place. Param2: Alpha string or expression that provides the sub-string to copy to Param1. Param3: First position in Param1 that receives the substring from Param2. Param4: Number of characters that are moved from Param2 to Param1, starting from the left most character of Param2. **Example:** @REP('12345', 'abcde', '3', '2') returns '12ab5'.
@RIGHT (Param1, Param2)	Returns a specified number of characters from an alpha string, starting with the character furthest right. The function right-justifies the string (removes trailing blanks) before commencing. **Parameters:** Param1: Alpha string from which the characters are taken. Param2: Number of characters to be retrieved, starting from the character furthest right. **Example:** @RIGHT('abcdefg ', '3') returns 'efg'.

String Function	Description
@RPAD (Param1, Param2)	Returns the specified parameter padded to the right using space characters as a padding string. The padding string is inserted to the right of the string until the length is reached. If padding is not performed (parameters are invalid or the length is smaller than the string length), the first parameter is returned. **Parameters:** Param1: String to be padded. Param2: Number that represents the length of the returned string. **Examples:** Square brackets are not part of the data: they are used to visually mark the length of the padded string. @RPAD ('constant','12') returns [constant]. @RPAD ('constant','3') returns [constant].
@RPAD (Param1, Param2, Param3)	Returns the specified parameter padded to the right using the specified padding string. The padding string is inserted to the right of the string until the length is reached. If padding is not performed (parameters are invalid or the length is smaller than the string length), the first parameter is returned. **Parameters:** Param1: String to be padded. Param2: Number that represents the length of the returned string. Paam3: String used for padding. **Examples:** Square brackets are not part of the data; they are used to visually mark the length of the padded string. @RPAD ('constant','12', 'L') returns [constantLLLL]. @RPAD ('constant','3', 'L') returns [constant].
@RTRIM (Param1)	Removes trailing white space (blanks, tabs, line feeds) from an alpha string or an alpha expression. **Parameter:** Param1: Input alpha string. **Example:** @RTRIM('John ') returns 'John'.

String Function	Description
@STRTOKEN (Param1, Param2, Param3)	Returns a string token from a delimited string. **Parameters:** Param1: Input string, delimited with tokens. Param2: Requested token index (numeric). Param3: Delimiter (can be an <XML> keyword). **Parameter Notes:** • Param3 can have more then one character as the delimiter. • An empty string is returned when a delimiter is not found, is empty, or if the input string is empty. • Every delimiter is counted for the index calculation (no repetition). **Example:** variable BA = abcd,cdef,ghik,lmnp, then @STRTOKEN(BA, '2', ',') returns cdef. **Note:** You can use valid <XML> element tags as delimeters. The XML format of a list can be a variable or URL. Content must consist of a <list><tag>Value</tag></list> set, where element names can be arbitrary but must be consistent.
@SUBSTR (Param1, Param2, Param3)	Extracts a specified number of characters (a sub-string) from an alpha string. **Parameters:** Param1: Input alpha string. Param2: Number that represents the starting position of the sub-string within Param1. Param3: Number of characters to be extracted (length of sub-string). **Example:** @SUBSTR('John', '3', '2') returns 'hn'.
@TRIM (Param1)	Removes leading and trailing white space (blanks, tabs, line feeds) from an alpha string or an alpha expression. **Parameter:** Param1: Input alpha string. **Example:** @TRIM(' John ') returns 'John'.

String Function	Description
@UPPER (Param1)	Converts a string to all uppercase. **Parameter:** Param1: Alpha string. **Example:** @UPPER('Pablo Picasso') returns 'PABLO PICASSO'.

Time Functions

The following table lists and describes the time functions that are available in iWay Transformer.

Time Function	Description
@ADD_DATE (Param1, Param2, Param3, Param4)	Performs a calculation on a date variable. Constructs a date out of an input date and 3 values to be added to that date: months, days, and years. The result is always a valid date. **Parameters:** Param1: Date (Format: MM/dd/yyyy). Param2: Number of months to add to Param1. Param3: Number of days to add to Param1. Param4: Number of years to add to Param1. **Example:** @ADD_DATE('01/01/1992', '1', '2', '2') returns 02/03/1994. **Note:** Zero values of the parameters are ignored.
@ADD_TIME (Param1, Param2, Param3, Param4)	Performs a calculation on a time variable. It constructs a time out of an input time and 3 values to be added to that time: hours, minutes, and seconds. The result is always a valid time. **Parameters:** Param1: Time value. Param2: Number of hours to add to Param1. Param3: Number of minutes to add to Param1. Param4: Number of seconds to add to Param1. **Example:** @ADD_TIME('12:00:00', '1', '2', '3') returns 13:02:03. **Note:** Zero values of the parameters are ignored.
@CUSTOMDATE()	Will be deprecated in future releases. Use @DATE.

Time Function	Description
@DATE (Param1)	Returns the system date. For more information, see *Date Pictures* on page A-34. **Parameter:** Param1: Date format. **Example:** If the system date is 01/28/1992, @DATE ('dd/MM/yyyy') returns 28/01/1992.
@DAY (Param1)	Returns the day portion of a date (a number between 1 and 31). **Parameter:** Param1: Date or date expression. **Example:** @DAY('01/28/1992') returns 28.
@DOW (Param1)	Returns the number of the day of the week, where Sunday is 1, Monday is 2, and so on. **Parameter:** Param1: (Format: MM/dd/yyyy). **Example:** @DOW('01/29/1992') (representing a Wednesday) returns 4.
@DSTR (Param1, Param2)	Returns specified date in the format specified by the second parameter. Specified date must be in the format: MM/dd/yyyy. For more information, see *Date Pictures* on page A-34. **Parameters:** Param1: String representing the date (Format: MM/dd/yyyy.) Param2: String that represents the picture mask (format) of the returned date string. **Example:** @DSTR('2/12/1998','MMMM dd, yyyy') returns February 12, 1998.

Time Function	Description
@DSTR (Param1, Param2, Param3)	Returns a specified date in the format specified by the third parameter. Specified date must be in the format specified by the second parameter. For more information, see *Date Pictures* on page A-34. **Parameters:** Param1: String that represents the date to be formatted. Param2: String that represents the picture mask of the Param1. Param3: String that represents the picture mask (format) of the returned date string. **Example:** @DSTR('2/12/1998','dd/MM/yyyy','MMMM dd, yyyy') returns December 02, 1998.
@DVAL (Param1, Param2)	Converts a date entered or stored as a character string to a numeric value. The numeric value represents the number of days elapsed since the day before the first day of the 1st century (01/01/01) until the date that is being converted. For more information, see *Date Pictures* on page A-34. **Parameters:** Param1: Character string or alpha expression that can be interpreted as a date (for example, '01/01/92', 'Jan 1, 1992'). Param2: Format for Param1; this parameter is required for the system to read and interpret the character string or expression. **Example:** @DVAL('01/01/92', 'MM/dd/yy') and @DVAL('Jan 1, 1992', 'MMM dd, yyyy') each return 727198.
@EOM (Param1)	Returns the date of the end of the month specified in the parameter. **Parameter:** Param1: Date or date expression. **Example:** @EOM ('05/05/93') returns 05/31/93.
@EOY (Param1)	Returns the date of the end of the year specified in the parameter. **Parameter:** Param1: Date or date expression. **Example:** @EOY ('10/05/93') returns '12/31/93'.

Time Function	Description
@HOUR (Param1)	Returns a number that represents the hours portion of a time value or a time expression. **Parameter:** Param1: Time value or time expression. **Example:** @HOUR('2:00:00') returns 2.
@MINUTE (Param1)	Returns a number that represents the minutes portion of a time value or a time expression. **Parameter:** Param1: Time value or time expression. **Example:** @MINUTE('2:35') returns 35.
@MONTH (Param1)	Returns the month portion of a date. **Parameter:** Param1: Date or date expression. **Example:** @MONTH('01/28/1992') returns 1.
@SECOND (Param1)	Returns a number that represents the seconds portion of a time value or a time expression. **Parameter:** Param1: Time value or time expression. **Example:** @SECOND('12:02:05') returns 5.
@SOM (Param1)	Returns the date of the start of the month specified in the parameter. **Parameter:** Param1: Date or date expression. **Example:** @SOM ('05/18/93') returns '05/01/93'.
@SOY (Param1)	Returns the date of the start of the year specified in the parameter. **Parameter:** Param1: Date or date expression. **Example:** @SOY ('10/05/93') returns '01/01/93'.
@TIME (Param1)	Returns the system time. For more information, see *Time Pictures* on page A-35. **Parameter:** Param1: Time format. **Example:** @TIME ('HH:mm:ss') returns 17:08:42.

Time Function	Description
@TSTR (Param1, Param2)	Converts a time to an alpha string, according to a picture mask. For more information, see *Time Pictures* on page A-35. **Parameters:** Param1: Time to be converted. Param2: Format of the resulting character string. **Example:** @TSTR ('14:30', 'HH:mm PM') returns '2:30 PM'. **Note:** A blank picture interprets the string as 'HH:mm:ss'.
@YEAR (Param1)	Returns the year portion of a date. **Parameter:** Param1: Date or date expression. **Example:** @YEAR('01/28/1992') returns 1992.

Numeric Pictures

The following table identifies the positional directives and mask characters for numeric pictures.

Symbol	Location	Meaning
0	Number	Digit.
#	Number	Digit, zero shows as absent.
.	Number	Decimal separator or monetary decimal separator.
-	Number	Minus sign.
,	Number	Grouping separator.
E	Number	Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix.
%	Prefix or suffix	Multiply by 100 and show as percentage.

Examples of numeric pictures are shown in the following table (the ^ symbol represents one space character).

Numeric Value	Picture	Resulting Numeric Value
1234.56	#,###.##	1,234.56

Numeric Value	Picture	Resulting Numeric Value
123456789.56	#,###.##	123,456789.56
-1234.56	N###,###.##C	^^-1,234.56
-1234.56	N######.##L	-1234.56^^
-1234.56	N######.##P*	-**1234.56
0	N######.##Z*	*********
-13.5	N##.##-DB;	DB13.50
45.3	N##.##+CR;	CR45.30
-13.5	N##.##-(,);	(13.50)
4055.3	$######.##	$^^4055.30

Date Pictures

The following table lists and describes the date symbols and shows an example of each.

Symbol	Meaning	Example	Presentation
G	Era designator	AD	Text
y	Year	1996	Number
M	Month in year	July & 07	Text & Number
d	Day in month	10	Number
E	Day in week	Tuesday	Text
D	Day in year	189	Number
F	Day of week in month	2 (2nd Wed. in July)	Number
w	Week in year	27	Number
W	Week in month	2	Number
‘	Escape for text	‘	Delimiter
‘ ‘	Single quote	‘	Literal
-	Separates month, day, and year.	12-24-86	Delimiter

Symbol	Meaning	Example	Presentation
/	Separates month, day, and year.	12/24/86	Delimiter

The typical date formats are dd/MM/yyyy (European), MM/dd/yyyy (American), or yyyy/MM/dd (Scandinavian). When you define the attribute Date for a variable, you must also select the format for the date item, as described in the following table. You can change this default picture and place in it any positional directives and mask characters you require.

The following table lists examples of date pictures, using the date of 21 March 1992 (the ^ symbol represents one space character).

Date Picture	Result and Notes
MM/dd/yyyy	03/21/1992
##/##/##	21/03/92 when an XML parser default is set to European. 03/21/92 when an XML parser is set to American.
MMMM^dd^yyyy	March^21^1992
MMM^dd, ^yyyy	Mar.^21^1992
EEEE^^-^7	Saturday^^^-^7
E^7	Mon^7

Time Pictures

The following table lists and describes the time symbols and shows an example of each.

Symbol	Meaning	Example	Presentation
h	Hour in am/pm (1-12)	12	Number
H	Hour in day (0-23)	0	Number
m	Minute in hour	30	Number
s	Second in minute	55	Number
-	Separates hours from seconds	1-22	1-22
:	Separates hours from seconds	1:22	1:22
S	Millisecond	978	Number

Symbol	Meaning	Example	Presentation
a	AM/PM marker	PM	Text
k	Hour in day (1-24)	24	Number
K	Hour in am/pm (0-11)	0	Number
z	Time zone	Pacific Standard Time	Text

The following table shows examples of time pictures and results.

Time Picture	Result	Comments
HH:mm:SS	08:20:00	Time displayed on 24-hour clock.
HH:mm:SS	16:40:00	Time displayed on 24-hour clock.
HH:mm PM	8:20 pm	Time displayed on 12-hour clock.
HH:mm PM	4:40 pm	Time displayed on 12-hour clock.
HH-mm-SS	16-40-00	Example of time separator of '-'.

APPENDIX B

Using Dictionary Files

Topics:

- Overview
- Composing EDI Dictionary Files
- Composing Swift Dictionary Files

A dictionary file is an XML representation of a transaction that is used by the Transformation Engine during an EDI or SWIFT transformation. Dictionary files are required whenever the transformation input or output data format is EDI or SWIFT.

This section describes how to compose EDI and SWIFT dictionary files.

Overview

iWay Transformer uses dictionary files to enable flexibility and ease of use when dealing with differences between types of EDI and SWIFT transactions. A dictionary file is an XML representation of an EDI or SWIFT transaction. The Transformation Engine can handle these transactions by using a dictionary file for each one, as this enables the Engine to essentially deal only with XML.

When EDI or SWIFT is your input format for an iWay Transformer transaction, the input is converted to XML using the structure defined in the appropriate dictionary file. Then, another transformation is performed into the desired output format. If EDI or SWIFT is your output type, your input file is transformed into XML, conforming exactly to the structure as defined in the specified dictionary file. Another conversion is made to the correct EDI transaction for your output using the same dictionary file.

To view a sample completed EDI dictionary file, navigate to

```
C:\Program Files\iway55\tools\transformer\samples\sample_projects\edi_x12
```

or to

```
C:\Program Files\iway55\tools\transformer\samples\sample_projects
\edi_hipaa
```

Default header information based on the respective EDI X12, EDI HIPAA, or EDIFACT standard can be provided for you, or you may choose to create your own header file.

Composing EDI Dictionary Files

A dictionary file and header are required for any iWay Transformer transformation in which EDI is the input or output data format.

Because a dictionary is an XML representation of a specific EDI transaction, you must create the dictionary to match exactly the structure of the transaction you are representing.

Note: The instructions in this topic assume that you are familiar with EDI, XML, and the specific EDI transaction with which you are working.

The structure you create is a combination of the following elements:

- EDI
- Segment
- Element
- CompositeElement
- ComponentElement

EDI dictionary headers are written using the same guidelines provided here for dictionary files, without the TransactionSet node.

A dictionary file has the following general node structure, which you can customize to conform exactly to the EDI structure you require.

```
<EDI>
    <TransactionSet>
       <Segment>
          <Element/>
          <Element/>
          ...
          <CompositeElement>
             <ComponentElement/>
             ...
          </CompositeElement>
          ...
       </Segment>
       <Loop>
           <Segment>
               <Element/>
               ...
               <CompositeElement>
                   <ComponentElement/>
                   ...
               </CompositeElement>
               ...
           ...
           <Loop>
               ...
           </Loop>
       </Loop>
      ...
    <TransactionSet>
</EDI>
```

XML Declaration

As with all XML documents, the top line of your file must be the XML declaration. You insert the following as your first line of text:

```
<?xml "1.0"?>
```

Everything that follows this line is your structure of nodes.

Comments

You may find it useful to insert comments into your dictionary file as you write it for later reference. Comments take the following form:

```
<!-- My comment here... -->
```

Comments do not affect your overall EDI structure. As with any XML document, a comment must begin with <!-- and end with -->. A comment can contain any sequence of characters except the double hyphen (--), which is only permitted to close the comment.

EDI

You place the EDI node at the top level of your structure's hierarchy. The EDI element must be given values for three attributes: Type, Version, and Standard. The Type attribute specifies the encoding that the text of your EDI transaction uses. For most purposes, this is ASCII.

For example, to define a HIPAA 4010 transaction type that uses the ASCII character set, the EDI opening tag would be defined as follows:

```
<EDI "ASCII" Version="4010" Standard="HIPAA">
```

A closing tag, </EDI>, is required at the end of the file. The EDI element always contains exactly one TransactionSet child.

TransactionSet

The TransactionSet node is always a child element of the EDI node. It must be given values for attributes ID and Name. Optionally, you can add an attribute Note whose value would contain information for your own reference.

For example, if you are building a dictionary file for the EDI HIPAA 276 transaction, Health Care Claim Status Request, the TransactionSet element would be defined as follows:

```
<TransactionSet ID="276" Name="Health Care Claim Status Request"
Note="Optional info...">
```

A closing tag, </TransactionSet>, is required at the end of the file just before the EDI closing tag. The TransactionSet element may contain as many Segment and Loop children as is required by your particular transaction.

Segment

A Segment node is always a child element of either a TransactionSet or Loop node. It must be given values for four attributes: ID, Name, Req, and MaxUse.

- ID: Is a two- or three-character value that identifies the Segment.
- Name: Specifies the Segment name.

- Req: Defines the requirement. It must be either 'M' (mandatory), or 'O' (optional).
- MaxUse: Sets the maximum allowed number of occurrences of the Segment, which is set to null if the Segment is allowed to exist an indefinite number of times.

Two optional attributes, Note and Type, exist for the Segment element. A value supplied to Note would contain information for your own reference. When multiple sibling loops exist in your transaction, Type is used to help the Transformation Engine differentiate between the loops. For more information, see *Segment Type Attribute* on page B-5.

For example, if you wish to add an ST segment, the Transaction Set Header, which is mandatory and can only occur once, your Segment element would be defined as follows:

```
<Segment ID="ST" Name="Transaction Set Header" Req="M" MaxUse="1">
```

A closing tag, , is required to close the segment. The Segment element may contain as many Element and CompositeElement children as is required by the transaction.

Segment Type Attribute

Type is only required when your dictionary file contains sibling loops that have the same first segment. For example, your dictionary file may have sibling loops 2000A and 2000B that both have a segment with ID="HL" as their first child segment. The Transformation Engine requires that the Type attribute be added to these HL segments to determine to which loop the different parts of the input data belong.

The Type attribute specifies a condition that is always true for the segment to which it refers and is unique to that loop. The condition could be that one of the segment elements does or does not exist, or that one of its elements has a certain value. If the condition is based on the existence of a child element value, it takes one of the following forms:

- Type="<ElementID>" - child element with specified ID has a value.
- Type="!<ElementID>" - child element with specified ID has a null value.

If the condition is based on the value of a child element, it takes one of the following forms:

- Type="<ElementID>=='<value>'" - child element with specified ID has given value.
- Type="<ElementID>!='<value>'" - child element with specified ID does not have given value.

For example, you may know that the third element of segment HL always has value '20' for loop 2000A and always has value '21' for loop 2000B. In this case, you would add Type="03=='20'" to segment HL in loop 2000A and Type="03=='21'" to segment HL in loop 2000B.

Loop

A Loop node is always a child element of either a TransactionSet node or another Loop node. It must be given values for the attributes, ID and MaxUse, which respectively identify the loop and define its maximum number of occurrences.

MaxUse is set to null if the loop is allowed to repeat an indefinite number of times. Optionally, you can add an attribute, Req, whose value would determine the requirement of the particular loop. Values for Req are either 'M' (mandatory) or 'O' (optional). If you do not include this attribute, the loop is considered optional.

For example, if you wish to create a loop 2000A that repeats as many times as required, the Loop element would be defined as follows:

```
<Loop ID="2000A" MaxUse="">
```

A closing tag, </Loop>, is required to close the loop. The Loop element may contain as many Segment and Loop children as is required by the transaction.

Element

An Element node is always a child element of a Segment node. It must be given values for six attributes: ID, Name, Req, Type, MinLength, and MaxLength. Optionally, you can add an attribute, Note, whose value contains information for your own reference.

- ID: A numerical value based on the Element position within its parent segment.

  Elements and CompositeElements are given ID values sequentially within their parent segment. For example, in a segment containing two Elements, one CompositeElement, and another Element, the respective ID values would be 01, 02, 03, and 04.

- Name: Specifies the Element name.

- Req: Defines the requirement.

  It must be either 'M' (mandatory), 'O' (optional), 'N' (not used), or a conditional requirement. For more information, see *Req: Conditional Requirement* on page B-7.

- Type: Specifies the type of the content that the Element node can hold. Permitted values are:

  - AN - string

    Element can contain a sequence of any characters from the basic or extended character sets.

  - DT - date

    Element value is either in format YYYYMMDD or YYMMDD, depending upon the MinLength and MaxLength attributes.

- ID - identifier

  Element contains a value from a pre-defined list of codes that is maintained by the ASC X12 Committee.

- Nn - numeric (when implemented takes on an integer value; for example, N0 and N2 both work).

  Element contains a numeric value with an implied decimal point position from the right. For example, a transmitted value of 789 when specified as numeric type N2 represents a value of 7.89. A leading minus sign (-) is used to indicate a negative value.

- R - decimal

  Element can contain an explicit decimal point (unlike Type Nn) in its value if the value is not an integer value. For example, a transmitted value of 7.89 when specified as type, R, represents a value of 7.89. A leading minus sign (-) is used to indicate a negative value.

- TM - time

  Element value is in general format HHMMSSd..d (where d are decimal seconds), and the precise format is determined by the MinLength and MaxLength attributes. For example, transmitted data of four characters denotes HHMM, and six characters denotes HHMMSS. Transmitted data of nine characters denotes HHMMSSddd.

- MinLength: Sets the minimum length permitted for the Element content.
- MaxLength: Sets the maximum length permitted for the Element content.

For example, if you wish to add a mandatory element as the first element in a segment that has alpha-numeric content with a minimum and maximum length of three and whose name is 'Transaction Set Identifier Code,' your Element would be defined as follows:

```
<Element ID="01" Name="Transaction Set Identifier Code" Req="M" Type="AN"
MinLength="3" MaxLength="3" Note="'ST01' used to select appropriate
transaction set definition"/>
```

No closing tag is required, but ensure you end all Element tags with />. Element elements cannot contain any children.

Req: Conditional Requirement

If the requirement of an Element or CompositeElement depends upon the outcome of a condition, then its Req attribute must be a conditional requirement. This requirement always evaluates to one of 'M' (mandatory), 'O' (optional), or 'N' (not used). A conditional requirement must take the form:

```
condition, trueReq, falseReq
```

A condition is comprised of a combination of terms, relational operators, and conditional operators. Terms are either sibling element IDs (for example, 04 or 12) or constant values (for example, 3 or foo).

Relational Operators

The following table lists and describes relational operators, gives an example of each, and indicates when an operator returns "true."

Operator	Meaning	Example	Returns true if
`==`	Is equal to	`01=='A'`	Sibling element 01 has a value equal to 'A'.
`!=`	Is not equal to	`02!='9'`	Sibling element 02 has a value not equal to '9'.
`>`	Is greater than	`03>'81'`	Sibling element 03 has a value greater than '81'.
`<`	Is less than	`04<'729'`	Sibling element 04 has a value less than '729'.

Conditional Operators: listed from highest to lowest precedence

The following table lists and describes conditional operators, gives an example of each, and indicates when an operator returns "true."

Operator	Meaning	Example	Returns true if
`<elementID>`	Sibling element with given ID exists	`01`	Value exists for sibling element 0.1.
`!`	Not	`!02`	Value does not exist for sibling element 02 (value is null).
`&amp;&amp;` See note*	And	`03&amp;&amp;11`	Values exists for both sibling elements 03 and 11.
`||`	Or	`08||26`	A value exists for either sibling element 08 or 26, or both values exist.

* && replaces the standard logic operator && due to the special character status given to the & character in XML.

You apply the trueReq requirement if the given condition evaluates to true. This can only take the value 'M', 'O', or 'N' and cannot be the same as the value given to falseReq.

You apply the falseReq requirement if the given condition evaluates to false. This can only take the value 'M', 'O', or 'N' and cannot be the same as the value given to trueReq.

Examples:

```
Req="04,M,O"
```

If a value exists for sibling element 04, then requirement is mandatory; otherwise requirement is optional.

```
Req="!05,M,O"
```

If the value of sibling element 05 is null, then requirement is mandatory; otherwise requirement is optional.

```
Req="02=='K'||03>'6'&amp;&amp;07||!09,M,O"
```

If sibling element 02 has value 'K', OR if sibling element 03 has value greater than '6' and a value for sibling element 07 exists, OR if the value of sibling element 09 is null, then requirement is mandatory; otherwise requirement is optional.

CompositeElement

A CompositeElement node is always a child element of a Segment node. It must be given values for the following attributes: ID, Name, and Req.

- ID: Is a numerical value based on the CompositeElement position within its parent segment.

  Elements and CompositeElements are given ID values sequentially within their parent segment. For example, in a segment containing two Elements, one CompositeElement, and another Element, the respective ID values would be 01, 02, 03, and 04.

- Name: Specifies the CompositeElement name.

- Req: Defines the requirement.

  It must be either 'M' (mandatory), 'O' (optional), or 'N' (not used).

For example, if you want to add a mandatory CompositeElement whose name is "Composite Medical Procedure Identifier" as the first child in a segment, your CompositeElement element would be defined as follows:

```
<CompositeElement ID="01" Name="Composite Medical Procedure Identifier" Req="M">
```

A closing tag, </CompositeElement>, is required to close this element. A CompositeElement can contain only ComponentElement children, but it can contain as many of these as is required by the transaction.

ComponentElement

A ComponentElement node is always a child element of a CompositeElement node. It must be given values for the following attributes: ID, Name, Req, Type, MinLength, and MaxLength.

- ID: Is a numerical value based on the ComponentElement position within its parent CompositeElement.

  For example, the first ComponentElement must have ID="01" and the fourth must have ID="04".

- Name: Specifies the ComponentElement name.
- Req: Defines the requirement.

  It must be either 'M' (mandatory), 'O' (optional), 'N' (not used) or a conditional requirement. For more information, see *Req: Conditional Requirement* on page B-7.

- Type: Specifies the type of content that the ComponentElement node can hold. The permitted values are:

  - AN - string

    ComponentElement can contain a sequence of any characters from the basic or extended character sets.

  - DT - date

    ComponentElement value is either in format YYYYMMDD or YYMMDD, depending upon the MinLength and MaxLength attributes.

  - ID - identifier

    ComponentElement contains a value from a predefined list of codes that is maintained by the ASC X12 Committee.

  - Nn - numeric (when implemented n takes on an integer value; for example, N0 and N2 both work)

    ComponentElement contains a numeric value with an implied decimal point n positions from the right. For example, a transmitted value of 789 when specified as numeric type N2 represents a value of 7.89. A leading minus sign (-) indicates a negative value.

  - R - decimal

    ComponentElement can contain an explicit decimal point (unlike Type Nn) in its value if the value is not an integer value. For example, a transmitted value of 7.89 when specified as type R represents a value of 7.89. A leading minus sign (-) indicates a negative value.

- TM - time

  ComponentElement value is in general format HHMMSSd..d (where d are decimal seconds), and the precise format is determined by the MinLength and MaxLength attributes. For example, transmitted data of four characters denotes HHMM, and six characters denotes HHMMSS. Transmitted data of nine characters denotes HHMMSSddd.

- MinLength: Sets the minimum length permitted for the ComponentElement content.
- MaxLength: Sets the maximum length permitted for the ComponentElement content.

For example, if you want to add an optional ComponentElement that has alpha-numeric content with minimum and maximum lengths of two, and whose name is "Procedure Modifier" as the sixth element in a CompositeElement, your ComponentElement would be defined as follows:

```
<ComponentElement ID="06" Name="Procedure Modifier" Req="O" Type="AN"
MinLength="2" MaxLength="2"/>
```

No closing tag is required, but ensure you end all ComponentElement tags with />. ComponentElement elements can not contain any children.

Composing Swift Dictionary Files

Each SWIFT transaction type must have a corresponding dictionary file to use the transaction with iWay Transformer. This topic describes how to create a dictionary for iWay Transformer to interpret SWIFT messages.

Note: The following instructions assume that you are familiar with SWIFT syntax, XML, and the specific SWIFT transaction with which you are working.

Starting SWIFT Messages

All SWIFT messages are surrounded by the parent root tag called <SWIFT> with an ID="SWIFT".

For example:

```
<SWIFT ID="SWIFT">
...
</SWIFT>
```

General Structure

SWIFT tags contain the following general structure:

1. Basic Header Block

2. Application Header Block

3. User Header Block

4. Text Block or Body

5. Trailer Block

1. Basic Header Block

The following is an example of basic header block.

```
<BASIC ID="BASIC" Req="M">
...
</BASIC>
```

The ID parameter can be a word or characters. We recommend that you use BASIC. The Req attribute can contain one of two values, representing the requirement of that block, either M (mandatory) or O (optional).

SWIFT Message Example

```
{1:     F         01          BANKBEBB     2222     123456}
        (a)       (b)         (c)          (d)      (e)
```

The following table defines the parts of the previous example of a SWIFT message.

(a)	Application ID - F = FIN, A = GPA or L = GPA (logins, etc.).
(b)	Service ID - 01 = FIN/GPA, 21 = ACK/NAK
(c)	LT address - 12 characters, must not have 'X' in position 9.
(d)	Session number - added by the CBT, padded with zeroes.
(e)	Sequence number - added by the CBT, padded with zeroes.

Writing a Basic Header (for a and b)

Note: (c), (d), and (e) are written in the same syntax.

A BASIC tag contains raw data. This data is represented in an Element tag that is written in the following syntax:

```
<BASIC ID="BASIC" Req="O">
      <Element ID="APPID" Name="Application ID" Req="M" Type="1!c"/>
      <Element ID="SRVID" Name="Service ID" Req="M" Type="2!n"/>
...
</BASIC>
```

- The ID parameter is a group of characters that references the Name parameter; this field is created by the author of the dictionary.

- The Name parameter specifies the raw data. For example, if raw data is 12345 and this is the account number of a client, then Name="Account Number".
- The Req parameter *must* be "M" for mandatory as specified by the user.
- The Type parameter tells the engine the types of characters and the amount of characters to expect.
- The final outcome for the two elements in the previous example is that APPID retrieves raw data "F", and SRVID retrieves raw data "01".

2. Application Header Block

The following is an example of application header block.

```
<APP ID="APP" Req="M">
...
</APP>
```

The ID parameter can be a word or characters. We recommend that you use APP. The Req attribute may contain one of two values, representing the requirement of that block, either 'M' (mandatory) or 'O' (optional).

SWIFT Message Example

```
{2:     I       100       BANKDEFFXXXX     U       3     003}
        (a)     (b)       (c)              (d)     (e)   (f)
```

The following table defines the parts of the previous example of a SWIFT message.

(a)	I = Input
(b)	Message Type
(c)	Receiver's address with an X in position 9 and padded with X's if there is no bank branch.
(d)	Message Priority: S = System, N = Normal, U = Urgent
(e)	Delivery Monitoring: 1 = Non Delivery Warning (MT010) 2 = Delivery Notification (MT011) 3 = Both Valid combinations of (d) and (e) are: U1 or U3, N2 or just N

(f)	Obsolescence Period: when a non-delivery notification is generated. Valid for U = 003 (15 minutes) Valid for N = 020 (100 minutes)

Writing an Application Header (a, b, and c)

Note: (d), (e), and (f) are written in the same syntax. This example is written for an input message where (a) is "I". For the best interpretation of the syntax, include two segment tags to represent both input and output. This ensures that an incoming message, whether input or output, is still be read.

An APP tag contains a Segment tag that is written in the following syntax:

```
<APP ID="APP" Req="O">
  <Segment ID="I" Name="Input">
  ...
  </Segment>
</APP>
```

- The ID parameter can be either "I" (input) or "O" (output).
- The Name parameter is Input if ID="I" or Output if ID="O".

A Segment tag contains an Element tag that is written in the following syntax:

```
<APP ID="APP" Req="O">
  <Segment ID="I" Name="Input">
     <Element ID="INOUT" Name="Input or Output flag" Req="M" Type="1!c"/>
        <Element ID="MSGTP" Name="Message Type" Req="M" Type="3!n"/>
        <Element ID="RCVAD" Name="Receive Address" Req="M" Type="12!c"/>
...
  </Segment>
</APP>
```

- The ID parameter is a group of characters that references the Name parameter; this field is created by the author of the dictionary.
- The Name parameter specifies the raw data. For example, if raw data is 12345 and this is the account number of a client, then Name="Account Number".
- The Req parameter *must* be "M" for mandatory as specified by the user.
- The Type parameter tells the engine the types and amount of characters to expect.
- The final outcome for the two elements in the previous example is that INOUT retrieves raw data "I", MSGTP retrieves raw data "100", and RCVAD retrieves "BANKDEFFXXXX".

3. User Header Block

The following is an example of user header block:

```
<USER ID="USER" Req="M">
...
</USER>
```

The ID parameter can be a word or characters. We recommend that you use USER. The Req attribute can contain one of two values, representing the requirement of that block, either 'M' (mandatory) or 'O' (optional).

SWIFT Message Example

```
{3:    {113:xxxx}       {108:abcdefgh12345678}      }
```

(a) (b)

The following table defines the parts of the previous example of a SWIFT message.

(a)	Optional banking priority code.
(b)	Message User Reference (MUR) used by applications for reconciliation with ACK.

Writing a User Header

A User tag contains a LOOP tag that is written in the following syntax:

```
<USER ID="USER" Req="O">
   <Loop ID="Info">
   ...
   </Loop>
</USER>
```

Note: The value of the ID parameter can be a word or characters specified by the author of the dictionary.

A LOOP tag contains Element tags that are written in the following syntax:

```
<USER ID="USER" Req="O">
   <Loop ID="INFO">
       <Element ID="CODE" Name="Generic code" Req="M" Type=""/>
       <Element ID="DATA" Name="Generic data" Req="M" Type=""/>
   </Loop>
</USER>
```

Note: For a loop in a user header, you can specify only two elements.

- The ID parameter is a group of characters that references the Name parameter; this field is created by the author of the dictionary.

- The Name parameter specifies the raw data, for example, if raw data is 12345 and this is the account number of a client, then Name="Account Number".
- The Req parameter *must* be "M" for mandatory as specified by the user.
- The Type parameter tells the engine the types and amount of characters to expect. In the previous example, a type parameter is not specified because any value can be used.
- The final outcome for (a) for the two elements in the previous example is that CODE retrieves raw data "113" and DATA retrieves "xxxx". For (b), the final outcome of the two elements is that CODE retrieves raw data "108" and DATA retrieves abcdefgh12345678.

4. Text Block or Body

The following is an example of user text block or body:

```
<TransactionSet ID="565" Name="Corporate Action Instructions">
...
</TransactionSet>
```

All text blocks are surrounded by a root tag called <TransactionSet>.

- The ID parameter is a group of characters that references the SWIFT message type number. For example, if you are creating a SWIFT message type 565, then the ID="565". This ID number *must* be the message type number specified by SWIFT.
- The Name parameter specifies the name given to the SWIFT message type referenced by 565. For example, 565 is the Corporate Action Instruction, therefore, Name="Corporate Action Instructions". For the name parameter, you can specify any name. However, we recommend that you use the name specified by the SWIFT message type.

Usage Rules for TransactionSet

The TransactionSet tags contain loop tags. Each sequence is a loop. A loop is written in the following syntax:

```
<Loop ID="A" Name="Sequence A" Req="M">
....
</Loop>
```

- The ID parameter is a unique ID specified in the SWIFT Format Specification table (can be anything, but the SWIFT Format Specification is recommended for consistency).
- The Name parameter is a name for the loop specified in the SWIFT Format Specification table (can be anything, but the SWIFT Format Specification name is recommended for consistency).
- The Req parameter *must* either be "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification table.

The Loop tags contain Segment and Element tags. Each SWIFT tag(s) surrounded by an arrow in the SWIFT Format Specification table is a loop. A loop is written in the following syntax:

```
<Loop ID="L_99A" Name="Loop of 99A" Req="M">
....
</Loop>
```

- The ID parameter is a unique ID created by the author of the dictionary.
- The Name parameter is a name for the loop created by the author of the dictionary.
- Req parameter *must* either be "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification table.

The Loop tags contain Segment and Element tags. Each SWIFT tag that has a lowercase letter in the SWIFT Format Specification table has a loop surrounding it. An example of a lowercase letter tag is 95a or 97a.

- 95a has four other options of 95P, 95Q, 95R, and 95S as shown in the Content column of the SWIFT Format Specification table.
- 97a has two other options of 97A and 97B as shown in the Content column of the SWIFT Format Specification table.

A loop is written in the following syntax:

```
<Loop ID="L_95a" Name="Loop of 95a" Req="M" Max="1">
....
</Loop>
```

- The ID parameter is a unique ID created by the author of the dictionary.
- The Name parameter is a name for the loop.
- The Req parameter *must* either be "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification table for that specific tag.
- The Max parameter in this situation is always set to "1".

  This means only looping once to validate whether the input SWIFT message is correct. This is optional, but is available in case the SWIFT message is wrong. For example, if SWIFT has 95A and another 95A immediately following. This is incorrect because only one instance of 95A is allowed.
- If this SWIFT tag is surrounded by an arrow in the SWIFT Format Specification table, then the Max parameter is not required.

The Loop tags contain Segment and Element tags. Each SWIFT tag is a segment tag. A Segment tag is written in the following syntax:

```
<Segment ID="20C" Name="Processing Reference" Req="M">
....
</Segment>
```

- The ID parameter *must* be the SWIFT tag specified in the SWIFT Format Specification table.
- The Name parameter is the name given to this SWIFT tag specified in the SWIFT Format Specification table (can be anything, but the SWIFT Format Specification name is recommend for consistency).
- The Req parameter is either "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification.

If a SWIFT tag is a Starting block, then the Segment tag is written in the following syntax:

```
<Segment ID="16R" Name="Starting Block" Req="M" Type="GENL">
....
</Segment>
```

- The ID parameter *must* be the SWIFT tag specified in the SWIFT Format Specification table. It is always "16R".
- The Name parameter is the name given to this SWIFT tag specified in the SWIFT Format Specification table. It is always "Starting Block" (can be anything, but the SWIFT Format Specification name is recommended for consistency).
- The Req parameter *must* either be "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification table.
- The Type parameter *must* be the characters given in the CONTENT column of the SWIFT Format Specification table.

If a SWIFT tag is an Ending block, then the Segment tag is written in the following syntax:

```
<Segment ID="16S" Name="Ending Block" Req="M" Type="GENL">
....
</Segment>
```

- The ID parameter *must* be the SWIFT tag specified in the SWIFT Format Specification table. It is always "16S".
- The Name parameter is the name given to this SWIFT tag specified in the SWIFT Format Specification table. It is always "Ending Block" (can be anything, but the SWIFT Format Specification name is recommended for consistency).
- The Req parameter *must* either be "O" for optional or "M" for mandatory, specified in the SWIFT Format Specification table.

- The Type parameter *must* be the characters given in the CONTENT column of the SWIFT Format Specification table.
- The Type parameter is usually the same as the Starting Block Type parameter. It signifies to the engine that this block is done.

If a SWIFT tag has a lowercase letter, there is a loop. The Segment tag for this case is written in the following syntax:

```
<Loop ID="L_97a" Name="Loop of 97a" Req="M" Max="1">
    <Segment ID="97A" Name="Account" Req="O">
    ....
    </Segment>
    <Segment ID="97B" Name="Account" Req="O">
    ....
    </Segment>
</Loop>
```

- The ID parameter *must* be the SWIFT tag specified in the SWIFT Format Specification table.
- The Name parameter is the name given to this SWIFT tag specified in the SWIFT Format Specification table (can be anything, but the SWIFT Format Specification name is recommend for consistency).
- The Req parameter is *always* "O" optional, regardless of whether the lowercase letter tag (for example, 97a) is mandatory or optional. The "M" and the "O" should be specified in the Loop tag (refer to loop that has a SWIFT tag of lowercase letters).

The Segment tags contain Element tags. SWIFT tags contain SWIFT messages with data. Data can be characters that signify an action or just raw data. This data is represented in an Element tag. An Element tag is written in the following syntax:

```
<Segment .....>
    <Element ID="QUALF" Name="Qualifier" Type=":4!c"/>
    <Element ID="REF" Name="Reference" Type="//16x"/>
</Segment>
```

- The ID parameter is a group of characters that references the Name parameter; this field is created by the author of the dictionary.
- The Name parameter is the name given to this SWIFT tag specified in the SWIFT Format Specification table (can be anything, but the SWIFT Format Specification name is recommended for consistency).
- The Type parameter tells the engine the types and amount of characters to expect.

Everything must be entered in the Type parameter, including /, //, :, and [ ]. For example, if the content is :4!c/16x//12c/[4!a][/12c], five Element tags are entered into the Type parameters accordingly:

(a) :4!c -> ... Type=":4!c"

(b) /16x -> ... Type="/16x"

(c) //12c -> ... Type="//12c"

(d) /[4!a] -> ... Type="/[4!a]"

(e) [/12c] -> ... Type="[/12c]"

- The Req parameter is not required, because the brackets ([]) in the Type parameter indicate optional components. Thus (d) and (e) are optional, that is, they are not required in the SWIFT message.
- If the Content column in the SWIFT Format Specification table has a constant in the Type parameter of the Element tag, you *must* replace it with the correct type accordingly, if the SWIFT Format Specification table does not provide a Type.

  For example, for 35B, the content is [ISIN1!e12!c] ->

  ```
  <Element .... Type="[4!c1!e12!c]" />
  ```

  where 4!C=ISIN
- If the Content column in the SWIFT Format Specification table has two or more types on separate lines, then you must specify this in the Type parameter, unless it is already specified in the SWIFT Format Specification table.

  The following is an example of a Type that is not specified. For the SWIFT tag 61, the content is:

  ```
  6!n[4!n]2a[1!a]15d1!a3!c16x[//16x]

  [34x]
  ```

  The Type for the second line is:

  ```
  Type="[1*34]"
  ```

  The characters 1* informs the engine of data on a new line of the SWIFT message that is part of the SWIFT tag 61.

  The following is an example of one that is already specified. For the SWIFT tag 35B, the content is:

  ```
  [ISIN1!e12!c]

  [4*35x]
  ```

The Type for the second line is:

```
Type="[4*35x]"
```

The characters 4*, already specified in the SWIFT Format Specification table, informs the engine that on the next four new lines of the SWIFT message, the data is part of the SWIFT tag 35B.

5. Trailer Block

The following is an example of trailer block:

```
<TRAILER ID="TRAILER" Req="M">
...
</TRAILER>
```

The ID parameter can be a word or characters. We recommend that you use TRAILER. The Req attribute can contain one of two values, representing the requirement of that block, either 'M' (mandatory) or 'O' (optional).

SWIFT Message Example

```
{5:   {MAC:12345678}        {CHK:123456789ABC}
```

(a) (b)

Writing a Trailer Block

A LOOP tag is written in the following syntax:

```
<Loop ID="INFO">
...
</Loop>
```

Note: The value of the ID parameter can be a word or characters specified by the author of the dictionary.

A LOOP tag contains Element tags that are written in the following syntax:

```
<TRAILER ID="TRAILER" Req="O">
    <Loop ID="INFO">
       <Element ID="CODE" Name="Generic code" Req="M" Type=""/>
       <Element ID="DATA" Name="Generic data" Req="M" Type=""/>
    </Loop>
</TRAILER>
```

Note: For a loop in a trailer header, you can specify only two elements.

- The ID parameter is a group of characters that references the Name parameter; this field is created by the author of the dictionary.

- The Name parameter specifies the raw data. For example, if raw data is 12345 and this is the account number of a client, then Name="Account Number".
- The Req parameter *must* be "M" for mandatory as specified by the author of the dictionary.
- The Type parameter tells the engine the types and amount of characters to expect. In the previous example, a type parameter is not specified because any value can be used.
- The final outcome for (a) of the two elements in the previous example is that CODE retrieves raw data "MAC" and DATA retrieves "12345678". For (b), the final outcome of the two elements is that CODE retrieves raw data "CHK", and DATA retrieves "123456789ABC".

APPENDIX C

Sample Transformations

Topics:

- Overview
- Transformation Principles
- Looping Example
- Context Sample
- Processing Column Defined Format Files

The following section provides transformation tutorials using iWay Transformer that demonstrate looping and context techniques.

Overview

Enterprise integration may require complex transformations of data. Some interfaces require transforming data from an RDBMS table format (rows and columns) into an XML hierarchical structure and back. Some of the transformations require conversion of hierarchical data into radically different hierarchies such as inversions of parent and child elements, aggregations, and possibly cross path mappings.

To accomplish these transformations, the principal behavior of iWay Transformer must be understood and alternative methods (for example, multiple transformations and XSLT) explored. This document considers a variety of scenarios that may occur and captures the default behavior of iWay Transformer when performing the transformation.

Transformation Principles

When reviewing the scenarios, several important points must be kept in mind. These principles reveal the processing methodology of iWay Transformer templates as they transform source documents into target documents.

Root Elements

By specification, well-formed XML documents contain one and only one outer element, referred to as the root element or tag. To ensure this condition, all target documents receive an initial root tag, Target_Root. This parent node is marked as a non-looping element.

iWay Transformer does support multi-root element documents, however, most XML tools cannot process them.

Parent Nodes

Parent nodes are XML elements that contain or wrap other elements. They can also contain data but are frequently used to indicate logically related elements and repeating (nested) elements.

Looping Example

Parent nodes that replicate for each set of constituent elements are looping. Properties of the parent node control this behavior.

Parent nodes can be marked as Loop (default), Loop (false), Loop (true), and Loop (aggregate) by manipulating the node properties. The default processing is for iWay Transformer to determine looping behavior. By selecting Loop property and selecting the loop option, the programmer can assume control of the looping strategy.

When looping is in effect, explicitly or by default, the parent node is repeated for each complete cycle of the constituent elements. This is sometimes confusing and possibly not the desired effect for the target document.

Procedure: How to Use Looping That is Set to True

To use looping that is set to true:

1. Create a new project.
2. Import a template file.
3. Load the Input Structure.
4. Select the *Mapping* tab.
   a. View the Output Items Structure.
   b. Right-click *b_out*.
5. Select *Properties*.
6. In the Properties window, change Loop to *True*.
7. View the output.
8. View the *b_out* parent.

   In the output, note that b_out is looping three times. This is because its loop is set to true and b_out is looping as many times as its children are looping.

   In this case, its children are looping three times because their mapping paths are looping in the input file:

```
b1_out child is mapped to path 'a/b/b1'
b2_out child is mapped to path 'a/b/b2'
```

Procedure: How to Use Looping That is Set to False

To use looping that is set to false:

1. Create a new project.
2. Import a template file.
3. Load the Input Structure.
4. Select the *Mapping* tab.
   a. View the Output Items Structure.
   b. Right-click *b_out*.
5. Select *Properties*.
6. In the Properties window, change Loop to *False*.
7. View the output.
8. View the *b_out* parent.

In the output you notice that b_out is not looping. This is because its loop is set to false, and b_out looping is disabled even though its children are looping.

In this case, its children are grouped under one b_out parent.

Procedure: How to Use Looping That is Set to Aggregate

To use looping that is set to aggregate:

1. Create a new project.
2. Import a template file.
3. Load the Input Structure.
4. Select the *Mapping* tab.
   a. View the Output Items Structure.
   b. Right-click *b_out*.
5. Select *Properties*.
6. In the Properties window, change Loop to *Agg*.
7. View the output.
8. View the *b_out* parent.

   In the output you note that b_out is looping twice. This is because its loop is set to aggregate, and b_out is looping only for unique parent blocks.

Reference: Sample Loop Input File

The following is a sample loop input file.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <a>
 - <b>
    <b1>value of input element a/b/b1</b1>
    <b2>value of input element a/b/b2</b2>
   </b>
 - <b>
    <b1>value of input element a/b/b1</b1>
    <b2>value of input element a/b/b2</b2>
   </b>
 - <b>
    <b1>value of input element a/b/b1</b1>
   </b>
</a>
```

Context Sample

The goal of the context example is to produce the exact same output as the input. That is why the input and output structures are the same.

Procedure: How to Use Context

To use context:

1. Create a new project.
2. Import a template file.
3. Load the Input Structure.
4. Select the *Mapping* tab.
5. View the Output Items Structure.

   Parent b has an element c with mapping path set to a/b/c.

   This path has two "b" loops in the input. These "b" loops are different because the values for the attribute, "name," are different.

   Currently, iWay Transformer does not recognize that the "b" loops are different. To the engine, the "b" loops are the same because their path a/b is the same.

   To avoid an unexpected result, you must specify that the input (a/b) is a different instance by setting "Context" in iWay Transformer.
6. Right-click *b_parent* and choose *Properties*.
7. Select Context to be *a/b*.

   With these settings you are giving iWay Transformer a "hint" when it produces the output tree structure. iWay Transformer is now aware which "c" belongs to which "b".

Reference: Sample Context Input File

The following is a sample context input file.

```
- <a>
  - <b name="b1">
     <c>value of input element a/b@name=b1/c is 1</c>
     <c>value of input element a/b@name=b1/c is 2</c>
    </b>
  - <b name="b2">
     <c>value of input element a/b@name=b2/c is 3</c>
     <c>value of input element a/b@name=b2/c is 4</c>
    </b>
</a>
```

Processing Column Defined Format Files

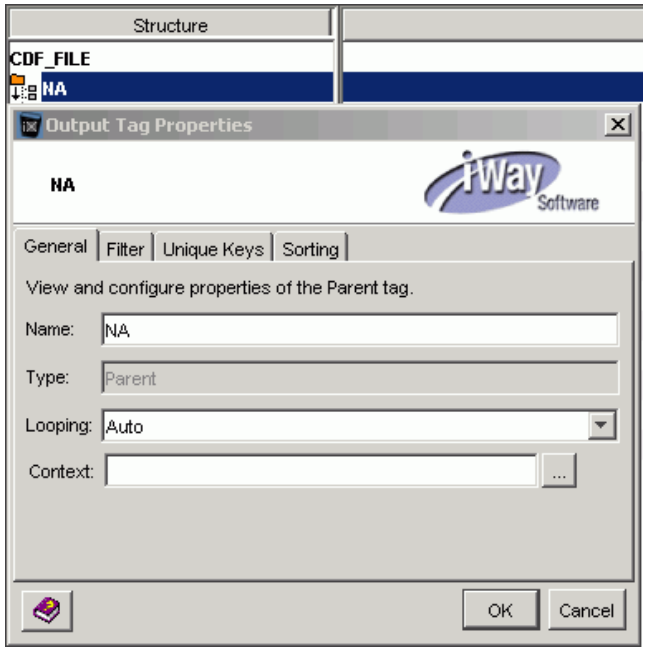
You can process Column Defined Format (CDF) files using iWay Transformer. Each line in an input CDF document contains multiple columns and has a predefined width, which means there are no delimiters between the columns. The following rules apply:

- A structure file is required to define the layout of the incoming document. This file uses a proprietary XML format and must be created manually.
- The input file may have an optional Header line, which may have multiple columns.
- Different types of records can be included, each of which has its own column definitions. When this is the case, each line must begin with a prefix record type identifier. This identifier can be of any length and contain any characters, for example, 10, 20, A, or B.

  Only records that match a prefix record type identifier are processed. A prefix record type identifier is not required if there is only one record type.

Note: To avoid receiving no output when generating CDF output, the Record Type field of the record layout cannot be left blank, for example, Type="".

The example files that are included have this field set to *NA*. It does not appear in the output if you uncheck the Visible checkbox. It is also important that Record Type is also marked as *NA* as shown in the following example:



Example: Processing a CDF File Containing no Header, One Record Type, and Three Columns

The following CDF file contains no header definition section. The type attribute of the record is blank because there is only one record type. Each column definition requires a start point and a length, for example:

- Column 1 starts at position 1 and contains 10 characters.
- Column 2 starts at position 11 and contains 29 characters.
- Column 3 starts at position 40 and contains 5 characters.

The total length of this record is 44 characters.

Note: In this example, an input document that contains less than 44 characters will cause the transformation to fail.

```
SAMPLE INPUT (orders1-Data.cdf): Each record contains 3 columns
~~~~~~~~~~~~~~~~~~~

 Fall1998  Full Zip Hooded Sweatshirt   10500

 Fall1998  North Star Polo              10501

 Spring1998Full Zip Hooded Sweatshirt   25500

 Spring1998North Star Polo              46500

 Summer1998Full Zip Hooded Sweatshirt   10505

 Summer1998North Star Polo              10506

 Winter1998Full Zip Hooded Sweatshirt   10510

 Winter1998North Star Polo              46500STRUCTURE [orders1-
CDFstruct.xml]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

<TransformCDFlayout>

    <RecordDetails>

    <RecordLayout Type="" StartOffset="1" Length="0">

     <Column Name="Quarter" StartOffset="1" Length="10"/>

     <Column Name="Description" StartOffset="11" Length="29"/>

     <Column Name="Order" StartOffset="40" Length="5"/>

     </RecordLayout>

  </RecordDetails>

</TransformCDFlayout>
```

The following syntax shows the output. Each line in the input CDF document is mapped to a REPORT element. Each REPORT element contains three child elements, one for each column.

```
OUTPUT~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<REPORTS>
    <Report>
        <Quarter>Fall1998  </Quarter>
        <Description>Full Zip Hooded Sweatshirt   </Description>
        <Order>10500</Order>
    </Report>
    <Report>
        <Quarter>Fall1998  </Quarter>
        <Description>North Star Polo              </Description>
        <Order>10501</Order>
    </Report>
    <Report>
        <Quarter>Spring1998</Quarter>
        <Description>Full Zip Hooded Sweatshirt   </Description>
        <Order>25500</Order>
    </Report>
    <Report>
        <Quarter>Spring1998</Quarter>
        <Description>North Star Polo              </Description>
        <Order>46500</Order>
    </Report>
    <Report>
        <Quarter>Summer1998</Quarter>
        <Description>Full Zip Hooded Sweatshirt   </Description>
        <Order>10505</Order>
    </Report>
    <Report>
        <Quarter>Summer1998</Quarter>
        <Description>North Star Polo              </Description>
        <Order>10506</Order>
    </Report>
    <Report>
        <Quarter>Winter1998</Quarter>
        <Description>Full Zip Hooded Sweatshirt   </Description>
        <Order>10510</Order>
    </Report>
    <Report>
        <Quarter>Winter1998</Quarter>
        <Description>North Star Polo              </Description>
        <Order>46500</Order>
    </Report>
</REPORTS>
```

Example: Processing a CDF File Containing a Header, Two Columns, One Record Type, and Three Columns

The data in the following example is identical to the first example, except that there is now a header definition section. The header line contains 2 columns. The first column identifies the report name and the second column identifies the store ID.

```
SAMPLE INPUT orders2-data.cdf:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 QuarterlyReportGAP123

 Fall1998  Full Zip Hooded Sweatshirt   10500

 Fall1998  North Star Polo              10501

...


Structure : Orders2-CDFstruct.xml ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

<TransformCDFlayout>

 <RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl"
Format="ASCII">

     <Column Name="ReportType" StartOffset="1" Length="15"/>

     <Column Name="StoreName" StartOffset="16" Length="6"/>

  </RecordHeader>

  <RecordDetails>

    <RecordLayout Type="" StartOffset="1" Length="0">

     <Column Name="Quarter" StartOffset="1" Length="10"/>

     <Column Name="Description" StartOffset="11" Length="29"/>

     <Column Name="Order" StartOffset="40" Length="5"/>

     </RecordLayout>

  </RecordDetails>

</TransformCDFlayout>
```

A record header definition has been added and the header row has two columns. There is still only one type of data record.

```
OUTPUT ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<REPORTS_WITH_HEADER>
    <REPORT_HEADER>
        <ReportType>QuarterlyReport</ReportType>
        <StoreName>GAP123</StoreName>
    </REPORT_HEADER>
    <REPORTS>
        <REPORT>
            <Quarter>Fall1998  </Quarter>
            <Description>Full Zip Hooded Sweatshirt   </Description>
            <Order>10500</Order>
        </REPORT>
        <REPORT>
            <Quarter>Fall1998  </Quarter>
            <Description>North Star Polo              </Description>
            <Order>10501</Order>
         </REPORT>

        ...

    </REPORTS>
</REPORTS_WITH_HEADER>
```

Example: Processing a CDF File Containing a Header, Two Columns, and Two Record Types

In the following example, the first record type has the prefix identifier set to 10 and contains three columns. It is the same as the records in the previous examples. The second record type has the prefix identifier set to 20 and contains two columns. For the sake of clarity in this example, the first column contains letters and the second contains numeric values, although this does not affect the structure definition.

```
INPUT [orders3-Data.cdf]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

QuarterSupplmntGAP123
20ABCDEFGHIJ1234567890
10Fall1998  Full Zip Hooded Sweatshirt   10500
20KLMNOPQRST9876543210
10Fall1998  North Star Polo              10501
10Spring1998Full Zip Hooded Sweatshirt   25500
10Spring1998North Star Polo              46500
10Summer1998Full Zip Hooded Sweatshirt   10505
10Summer1998North Star Polo              10506
10Winter1998Full Zip Hooded Sweatshirt   10510
10Winter1998North Star Polo              46500
```

A header line is included and there are two types of records identified with the prefixes 10 and 20. In addition, there are eight records set to 10 and two records set to 20.

No relationship between different record types can be implied. For example, do not assume that a record type of 20 is the parent of the following record types set to 10.

```
<!--Sample CDF Structure -->
<TransformCDFlayout>
  <RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl"
Format="ASCII">
     <Column Name="ReportType" StartOffset="1" Length="15"/>
     <Column Name="StoreName" StartOffset="16" Length="6"/>
  </RecordHeader>
  <RecordDetails>
    <RecordLayout Type="10" StartOffset="1" Length="2">
      <Column Name="Quarter" StartOffset="3" Length="10"/>
      <Column Name="Description" StartOffset="13" Length="29"/>
      <Column Name="Order" StartOffset="42" Length="5"/>
    </RecordLayout>
    <RecordLayout Type="20" StartOffset="1" Length="2">
      <Column Name="code-letters" StartOffset="3" Length="10"/>
      <Column Name="code-numbers" StartOffset="13" Length="10"/>
    </RecordLayout>
  </RecordDetails>
</TransformCDFlayout>
OUTPUT ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<ORDER-DATA>
    <HEADER>
        <ReportType>QuarterSupplmnt</ReportType>
        <STORE>GAP123</STORE>
    </HEADER>
    <_10_REPORT_RECORD>
        <QUARTER>Fall1998  </QUARTER>
        <Description>Full Zip Hooded Sweatshirt   </Description>
        <Order>10500</Order>
    </_10_REPORT_RECORD>
```

```
<_10_REPORT_RECORD>
    <QUARTER>Fall1998  </QUARTER>
    <Description>North Star Polo              </Description>
    <Order>10501</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
    <QUARTER>Spring1998</QUARTER>
    <Description>Full Zip Hooded Sweatshirt   </Description>
    <Order>25500</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
    <QUARTER>Spring1998</QUARTER>
    <Description>North Star Polo              </Description>
    <Order>46500</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
    <QUARTER>Summer1998</QUARTER>
    <Description>Full Zip Hooded Sweatshirt   </Description>
    <Order>10505</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
    <QUARTER>Summer1998</QUARTER>
    <Description>North Star Polo              </Description>
    <Order>10506</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
    <QUARTER>Winter1998</QUARTER>
    <Description>Full Zip Hooded Sweatshirt   </Description>
    <Order>10510</Order>
</_10_REPORT_RECORD>
<_10_REPORT_RECORD>
```

```
            <QUARTER>Winter1998</QUARTER>
            <Description>North Star Polo              </Description>
            <Order>46500</Order>
        </_10_REPORT_RECORD>
        <_20_SUPPLEMENTAL_RECORD>
            <code-letter>ABCDEFGHIJ</code-letter>
            <code-number>1234567890</code-number>
        </_20_SUPPLEMENTAL_RECORD>
        <_20_SUPPLEMENTAL_RECORD>
            <code-letter>KLMNOPQRST</code-letter>
            <code-number>9876543210</code-number>
        </_20_SUPPLEMENTAL_RECORD>
</ORDER-DATA>
```

The Transform Mapping defines the output to consist of the following XML elements:

- **<HEADER>**

  Contains two header columns.

- **<_10_REPORT_RECORD>**

  Contains three 10 record columns.

- **<_20_SUPPLEMENTAL_RECORD>**

  Contains two 20 record columns.

As a result, the header data is first, followed by all the record types set to 10 and then all record typese set to 20. This is regardless of the order in which they are input, for example, the first record in the sample data was set to 20.

The fact that different record types, for example 10 and 20 are generated in the exact order they are input cannot be changed. However, all records within the same type are generated in the input order.

Removing an Empty Header Line From the CDF Output

By default, an empty line is included in the header section of the CDF output, even if you do not map anything into it in your transformation.

Deleting this node on the Mapping screen results in an error. If you delete the header element from the structure file, it still appears on the Mapping screen. However, you can filter it out.

Procedure: How to Remove an Empty Header Line From the CDF Output

To remove an empty header line from your CDF output, you need to specify a condition to receive CDF output without a header.

1. Set the Filter Subtree option property for the Header node.
2. For the condition, enter the following:

   ```
   RootElementName == '1'
   ```

   Assuming this condition is always false, the empty header line is filtered out from the CDF output.

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

Mail: Documentation Services - Customer Support
Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898

Fax: (212) 967-0460

E-mail: books_info@ibi.com

Web form: http://www.informationbuilders.com/bookstore/derf.html

Name:_____

Company:_____

Address:_____

Telephone:_____Date:_____

E-mail:_____

Comments:

Reader Comments

Information Builders, Two Penn Plaza, New York, NY 10121-2898 **(212) 736-4433**